PCT/JP2004/013855

24. 9. 2004

# 日本国特許庁
# JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.



出願年月日
Date of Application: ２００３年　９月２２日

出願番号
Application Number: 特願２００３－３３０７７２

[ST. 10/C]: [ＪＰ２００３－３３０７７２]

出願人
Applicant(s): カテナ株式会社

**PRIORITY DOCUMENT**
SUBMITTED OR TRANSMITTED IN
COMPLIANCE WITH
RULE 17.1(a) OR (b)



２００４年１１月　４日

特許庁長官
Commissioner,
Japan Patent Office

小川洋

出証番号　出証特２００４－３０９９１９１

【書類名】　特許願
【整理番号】　DCT03034
【特記事項】　特許法第３６条の２第１項の規定による特許出願
【提出日】　平成15年 9月22日
【あて先】　特許庁長官殿
【国際特許分類】　F01B 23/12
F01C 17/00

【発明者】
【住所又は居所】　カナダ、ケベック州、Ｇ１Ｋ　７Ｐ４、ラバール大学コンピュータ科学学部内
【氏名】　モハメド・メジリ

【発明者】
【住所又は居所】　カナダ、ケベック州、Ｇ１Ｋ　７Ｐ４、ラバール大学コンピュータ科学学部内
【氏名】　ベシール・クタリ

【発明者】
【住所又は居所】　岩手県岩手郡滝沢村滝沢字巣子１５２－５２　岩手県立大学ソフトウェア情報学部内
【氏名】　藤田　ハミド

【特許出願人】
【識別番号】　396023362
【氏名又は名称】　カテナ株式会社
【氏名又は名称原語表記】　ＣＡＴＥＮＡ　ＣＯＲＰＯＲＡＴＩＯＮ
【代表者】　小宮　善継
【国籍】　日本

【特許出願人】
【識別番号】　599086238
【氏名又は名称】　ソフトウェア生産技術研究所株式会社
【氏名又は名称原語表記】　ＴＨＥ　ＩＮＳＴＩＴＵＴＥ　ＯＦＣＯＭＰＵＴＥＲ　ＢＡＳＥＤ　ＳＯＦＴＷＡＲＥ　ＭＥＴＨＯＤＯＬＯＧＹ　ＡＮＤ　ＴＥＣＨＮＯＬＯＧＹ
【代表者】　根来　文生
【国籍】　日本

【代理人】
【識別番号】　100110559
【弁理士】
【氏名又は名称】　友野　英三
【電話番号】　0422-39-3200
【ファクシミリ番号】　0422-31-9444

【手数料の表示】
【予納台帳番号】　164782
【納付金額】　35,000円

【提出物件の目録】
【物件名】　外国語特許請求の範囲　1
【物件名】　外国語明細書　1
【物件名】　外国語図面　1
【物件名】　外国語要約書　1

【書類名】外国語特許請求の範囲

A
software generating method comprising:
A step for defining a software requirements by use of Lyee methodology;
A step for arranging the requirements defined by use of Algebra process;
A step for inputting the arranged requirements into Pallets and Scenario Function of the Lyee methodology to get the source code; and
A step for running the gotten source code until the original requirements are met.

【書類名】外国語明細書

# Lyee Methodology: A Formalization using Process Algebra

**Abstract.** Over the last years, various methodologies and techniques have been elaborated and proposed to improve one or many aspects related to the software development life cycle. However, despite the great effort in this research field, the production of clearly understood and modifiable systems still an ambitious goal and far from reached. This is due, on one hand, to the complexity and the subtlety of software themselves and, on the other hand, to the limitations of the current methodologies. Recently, a new and very promising methodology, called Lyee, has been proposed. Intended to deal efficiently with a wide range of software problems related to different field, Lyee allows the development of software by simply defining their requirements.

Nevertheless, since both the semantics of Lyee generated software together with the process of automatic generation of software from requirements are described using informal language, difficulties and confusions may arise when trying to understand and study this methodology.

The main purpose of this paper is to formalize, using a process algebra, the process of automatic generation of software together with the semantics of Lyee generated software. Actually, process algebra naturally support many concept of the Lyee methodology and consequently formalize them simply and elegantly. It offers the Lyee methodology an abstract machine more suitable than the Von-Newman one. In fact, this new abstract machine consider a program as chemical solution when molecules (different vectors of the lyee methodology) interact together to reach a collective goal.

**keywords:** *Lyee Methodology, Process Algebra, Operational Semantics, Bisimulation.*

## 1 Introduction

Producing easily and quickly software with hight qualities is the basic concern of the software development research field. Over the last years, various methodologies and techniques have been elaborated and proposed to improve one or many aspects related to the software development life cycle. However, despite the great effort in this research field, the production of clearly understood and modifiable systems still an ambitious goal and far from reached. This is due, on one hand, to the complexity and the subtlety of software themselves and, on the

other hand, to the limitations of the current methodologies. In fact, almost all of the proposed methodologies fail to produce clearly understood and modifiable systems and their use is still considered to be an activity accessible only to specialists with a large array of competencies, skills, and knowledge. This, in turn, leads to highly paid personal, high maintenance costs, and extensive checks needing to be performed on the software. For these reasons, companies are likely to welcome any new methodology promising demonstrable improvement in the software development cycle.

Recently, a new and very promising methodology, called Lyee [6, 7, 8, 9] (governementaL methodologY for softwarE providencE), has been proposed. Intended to deal efficiently with a wide range of software problems related to different fields, Lyee allows the development of software by simply defining its requirements. More precisely, a developer has only to provide words, calculation formulae, calculation conditions (preconditions) and layout of screens and printouts, and then leaves in the hands of the computer all subsequent troublesome programming process (e.g., control logic aspects). Despite its recency, the results of the use of Lyee have shown tremendous potential. In fact, compared to conventional methodologies, development time, maintenance time and documentation volume can be considerably reduced by using Lyee (as much as 70 to 80%) [7]. In [4], we proposed some classical static analysis techniques in order to improve many aspects of this methodology, in particular Lyee requirements.

Nevertheless, since both the semantics of Lyee generated software together with the process of automatic generation of software from requirements are described using informal language, difficulties and confusions may arise when trying to understand and study this methodology. In addition, the ideas behind this methodology are described using a sequential langage which is not appropriate to support them. In fact, a Lyee generated software is basically made of small components (called vectors in Lyee terminology), where each one has an atomic goal, that collaborate together by interaction in order to produce the desired results (global goal). It is, however, commonly known that a process algebra which naturally supports concurrency and communication is the appropriate langage to specify concurrent components and therefore it will provides a formal and worthy foundation to support the Lyee methodology concepts. Among other benefic consequences of using process algebra with the Lyee methodology, many components (complete vectors or parts of vectors) of the actual version of Lyee, which their presence was forced by the sequential langage used to describe it, will no longer be necessary since they are naturally supported by process algebra. Actually, synchronization vector, most of the routing vectors, command vector, etc., are no longer needed. Thus, we obtain a formal description of Lyee that is simple and that produce program shorter and consuming less time and memory. Besides, this formal description will be an inevitable start point for many interesting analysis of diverse aspects of this methodology. For instance, to optimize the Lyee generated software or the software generation processes, we need a formal proof that the optimized program is *equivalent* to its original version. Formal equivalence checking or more generally model checking can be elegantly achieved when using process algebra.

In this paper, we first define a formal process algebra, called Lyee-calculus, that easily and naturally supports the basic concepts of the Lyee methodology. In fact, this calculus can be seen as an abstract machine which is more suitable to support the Lyee methodology concepts than the Von Newman one. This machine considers a program as a set of molecules that interact together to produce a final result. Secondly, we show how this calculus can formalize and simplify both the lyee vectors together with the whole software generation

process. Actually, we have formalize the whole process of the automatic generation of Lyee software generation.

The remainder of this paper is organized as follows. In Section 2, we define the syntax and the semantics of the Lyee-calculus. In Section 3, we give a short and technical introduction to the Lyee methodology and an overview of its formalization using the Lyee-calculus. In Section 4, we propose a detailed and complete formalization of the Lyee methodology using our calculus. In Section 5, we present one case study that concretely shows both how a program is automatically generated from simple requirement together with a step-by-step description of the execution of the generated program. Finally, Section 6 provides concluding remarks on this work, and discusses some future work

## 2 Lyee-Calculus

Process algebra is a formal description technique for complex computer systems, especially those involving communicating, concurrently executing components [1, 3, 5]. Process algebra is generally referred as calculus in the literature since it involves a variety of mathematical and logic concepts (concurrency theory, operational semantics, complexity theory, logic, etc.) that goes beyond the process algebra field.

In this section, we give the syntax and the semantics of a new calculus, called Lyee-calculus, that we have specially define to formalize the Lyee methodology.

### *2.1 Syntax*

Lyee-calculus programs are systems composed of independent and parallel processes that communicate using hand-shake technique on named channels. The channel can be *restricted* so that only some process can communicate on them. A process can send a value $v$ through a channel $z$ by doing the action $[z!e]$ ($v$ corresponds to the valuation of $e$). Similarly, a process can receive a value from a channel by doing the action $[z?e]$. A processus can also performs a silent action $\tau$. This special action, intended to represent internal behavior of a system such as synchronization between processes, is also useful to capture indeterminism within process evolution.

The syntax of processes is presented in Table 1. The intuitive meaning of each syntactic construction is as follows:

- *Sequence*: $[K].P$, where $K$ is a set of actions, is a process that has to perform all the action in $K$ and then behaves as the process $P$. The order in which the actions in $K$ have to be executed is not important. Notice that for the sake of simplicity we write $[\kappa_1, \ldots, \kappa_n]$ instead of $[\{\kappa_1, \ldots, \kappa_n\}]$.

- *Parallel composition*: $P \mid Q$ behaves as processes $P$ and $Q$ running in parallel. Each one of them may interact with the other on channels known to both, or with the outside word (environment or the end-user) independently from the other. When two processes synchronise on the same channel, the whole process will perform an action $\tau$ and behaves as the remaining process. For instance, let's take the following process:

$$P = P1 \mid P2$$

Table 1: Syntax of Process Algebra.

| | | | |
|---|---|---|---|
| $P, Q$ | ::= | | (Processes) |
| | \| | $[K].P$ | (Sequence) |
| | \| | $P \mid Q$ | (Parallel composition) |
| | \| | $P + Q$ | (Choice) |
| | \| | $P \rhd Q$ | (Guarded choice) |
| | \| | $P/L$ | (Restriction) |
| | \| | $A(\vec{X}) \stackrel{def}{=} P$ | (Definition) |
| | \| | $nil$ | (nil process) |
| $K, K_1, K_2$ | ::= | | (Set of actions) |
| | \| | $\{\kappa\}$ | (Single action) |
| | \| | $K_1 \cup K_2$ | (Set union) |
| $L, L_1, L_2$ | ::= | | (Set of channels) |
| | \| | $\emptyset$ | (Empty set) |
| | \| | $\{i\}$ | (Single channel) |
| | \| | $L_1 \cup L_2$ | (Set union) |
| $\kappa$ | ::= | | (Action) |
| | \| | $i!e$ | (Send) |
| | \| | $i?e$ | (Receive) |
| | \| | $\tau$ | (Silent action) |
| $i, j$ | | | (Channel) |
| $e$ | | | (Arithmetic expression) |

with $P1$ and $P2$ defined as follows:

$$P1 \stackrel{def}{=} [i!4].P1'$$
$$P2 \stackrel{def}{=} [i?x].P2'$$

then, process $P$ performs a silent action $\tau$, indicating a synchronization between processes $P1$ and $P2$ which actually consist in the transmission of the value 4 from $P1$ to $P2$, and behaves as the remain process $P1' \mid P2'$. Semantically, we will note this transition as follows:

$$P \xrightarrow{\tau} P1' \mid P2'$$

The relation $\xrightarrow{\kappa}$ will be formally defined later within this section.

- *Choice*: $P + Q$ behaves as $P$ or as $Q$. The choice is deterministically (made by the environment) if both $P$ and $Q$ do not begin with a silent action, otherwise the choice is not deterministically.
- *Guarded choice*: $P \rhd Q$ is the process that behaves as $P$ until process $Q$ is activated. Whenever the latter is activated, the former is stopped and cleared from memory.

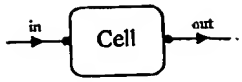



Figure 1: Cell formalized as a process.

- *Restriction*: $P/L$ is the process that behaves as $P$ except that it can communicate with the environment using channels given in $L$ only.
- *Definition*: $A(\overrightarrow{X}) \stackrel{def}{=} P$ is a defining equation where $A$ is a process identifier, $\overrightarrow{X}$ are variables (parameters), and $P$ may recursively involve $A$.
- *nil process*: $nil$ is the process that cannot perform any action.

### 2.2 Example

In this section, we show how to model a cell of memory as a process that interacts with its environment through its communication channels. As shown by Fig. 1, we consider that the cell has two ports (channels) of communication, $in$ and $out$. The basic task of this cell is to infinitely wait a value on channel $in$ and to make it available on channel $out$. The same value may be outputted as mush as necessary until a new value is imputed.

We write the process cell $C^x(v)$, meaning that the memory cell $x$ holds the value $v$, as follows:

$$C^x(v) \stackrel{def}{=} [in^x?y].C^x(y) + [out^x!v].C^x(v)$$

By capturing a memory cell as a process, we can add intelligence to it. For instance, we can write a cell that it does not allow the access to its content until it will be initialized. This smart cell can be defined as follows:

$$Cell(x) \stackrel{def}{=} [in^x?y].C^x(y)$$

Now, giving two processes defined as follows:

$$\begin{aligned} P1 &\stackrel{def}{=} [in^x!5].nil \\ P2 &\stackrel{def}{=} [out^x?y].nil \end{aligned}$$

it is easy to write a program where these two processes communicate through a cell $x$:

$$P1 \mid Cell(x) \mid P2$$

The Fig. 2 shows the interaction between all the involved processes. Here are the different steps of that program execution:

$$\begin{aligned} P1 \mid Cell(x) \mid P2 &= [in^x!5].nil \mid [in^x?y].C^x(y) \mid [out^x?y].nil \\ &\xrightarrow{\tau} C^x(5) \mid [out^x?y].nil \\ &= ([in^x?y].C^x(y) + [out^x!5].C^x(5)) \mid [out^x?y].nil \\ &\xrightarrow{\tau} C^x(5) \end{aligned}$$


Figure 2: Example of interacting processes.

### 2.3 *Semantics*

Hereafter, we give the formal semantics of the Lyee-calculus. This semantics is defined by the interaction relation $\xrightarrow{\kappa}$, where $P \xrightarrow{\kappa} Q$ means that there is a reaction amongst the sub-processes of $P$ such that the whole can execute the atomic action $\kappa$ and becomes $Q$. The interaction relation is inspired by the Chemical Abstract Machine of Berry and Boudol [2]. In this model, a process is considered as a chemical solution of molecules waiting to interact. To formally introduce the relation $\xrightarrow{\kappa}$, we need to define the following notions:

- $\leadsto$ is a relation that simplify processes by eliminating the *nil* process as follows:

$$\begin{array}{lll} P \mid nil \leadsto P & nil + P \leadsto P & nil \rhd P \leadsto P \\ nil \mid P \leadsto P & P + nil \leadsto P & P \rhd nil \leadsto P \end{array}$$

$$nil/L \leadsto nil$$

- We denote by $[\![e]\!]$ the set of values containing the result of the evaluation of the expression $e$. If $e$ is a variable, it evaluation will be its domain (int, real, etc.). For the sake of simplicity, we consider that all variables belong to the real set.

- $\kappa_!$ is the name of the channel used by the action $\kappa$, i.e:

$$\begin{array}{rcl} \tau_! & = & \emptyset \\ (a!e)_! & = & (a?e)_! = \{a\} \end{array}$$

Now, we let the interaction reaction $\xrightarrow{\kappa}$ be the least relation that satisfies the rules given by Table 2.

## 3 Informal Formalization of Lyee Requirements

This section presents an overview of how the Lyee methodology generates software from simple user requirements. Besides, through this overview we progressively introduce how one can simply and efficiently formalize this methodology using the Lyee-calculus. The complete formalization of the process of software generation of the lyee methodology will be given in the next section.

### 3.1 *Lyee requirements*

Within the Lyee methodology, requirements are given in a declarative way as a set of *statements* containing words together with their definitions, their calculation conditions and their

Table 2: Semantics of Lyee-Calculus

$$(R_{\leftarrow})\ \frac{P \xrightarrow{\kappa} Q' \quad Q' \leftarrow\!\!\!\shortmid\ Q}{P \xrightarrow{\kappa} Q}$$

$$(R_!)\ \frac{v \in [\![e]\!]}{[z!e].P \xrightarrow{z!v} P} \qquad (R_?)\ \frac{v \in [\![e]\!]}{[z?e].P \xrightarrow{z?v} P[v/e]} \qquad (R_\tau)\ \frac{v \in [\![e]\!]}{[\tau].P \xrightarrow{\tau} P}$$

$$(R_{[\,]})\ \frac{[K_1].P \xrightarrow{\kappa} P'}{[K_1 \cup K_2].P \xrightarrow{\kappa} [K_2]P'}\ K_2 \neq \emptyset$$

$$(R^l_+)\ \frac{P \xrightarrow{\kappa} P'}{P+Q \xrightarrow{\kappa} P'} \qquad (R^r_+)\ \frac{Q \xrightarrow{\kappa} Q'}{P+Q \xrightarrow{\kappa} Q'}$$

$$(R^\tau_\triangleright)\ \frac{P \xrightarrow{z?v} P' \quad Q \xrightarrow{z!v} Q'}{P \triangleright Q \xrightarrow{\tau} Q'} \qquad (R^l_\triangleright)\ \frac{P \xrightarrow{\kappa} P'}{P \triangleright Q \xrightarrow{\kappa} P' \triangleright Q} \qquad (R^r_\triangleright)\ \frac{Q \xrightarrow{\kappa} Q'}{P \triangleright Q \xrightarrow{\kappa} Q'}$$

$$(R^\tau_|)\ \frac{P \xrightarrow{z?v} P' \quad Q \xrightarrow{z!v} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad (R^l_|)\ \frac{P \xrightarrow{\kappa} P'}{P \mid Q \xrightarrow{\kappa} P' \mid Q} \qquad (R^r_|)\ \frac{Q \xrightarrow{\kappa} Q'}{P \mid Q \xrightarrow{\kappa} P \mid Q'}$$

$$(R_=)\ \frac{P[\vec{Y}/\vec{X}] \xrightarrow{\kappa} P'}{A(\vec{Y}) \xrightarrow{\kappa} P'}\ A(\vec{X}) = P \qquad (R_/)\ \frac{P \xrightarrow{\kappa} P'}{P/L \xrightarrow{\kappa} P/L}\ \kappa_1 \in L$$

attributes (input/output, types, security attributes, etc.). Table 3 gives an example of Lyee requirements.

Table 3: Lyee Requirements.

| Word | Definition | Condition | IO | Type | Security |
|---|---|---|---|---|---|
| | | ⋮ | | | |
| a | b+c | b*e>2 | OF | int | secret |
| c | | | IS | float | public |
| b | 2*c+5 | c>0 | OS | float | public |
| e | | | IS | float | public |
| | | ⋮ | | | |

The requirements given in that table correspond intuitively, in a traditional programming language, to the code given in Table 4.

Table 4: Statement Code.

| Statement | Code |
|---|---|
| $s_a$ | if b*e>2<br>then a:=b+c; output(a);<br>endif |
| $s_c$ | input(c); |
| $s_b$ | if c>2<br>then b:=2*c+5; output(b);<br>endif |
| $s_e$ | input(e); |

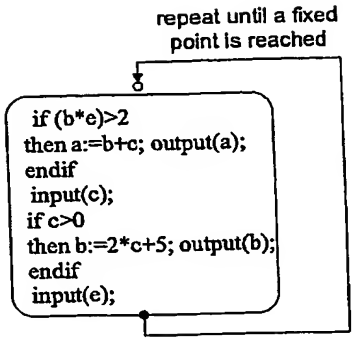



Figure 3: Requirement Execution.

Within the Lyee methodology, the user does not need to specify the order (control logic) in which these definitions will be executed. As shown in Table 4, despite the fact that the definition of word $a$ uses word $b$, statement $s_b$ is listed after the statement $s_a$. As explained in the sequel, from these requirements, and independent of the order of statements, Lyee is able to generate code that computes all the defined words. This simple idea has, as shown in [6, 7, 8, 9], multiple beneficial consequences on the different steps of software development. In fact it allows us to begin developing software even with incomplete requirements. Moreover, the user does not need to deal with control logic as with more classical methodologies. The control logic part of the software will be, within the Lyee methodology, automatically generated reducing consequent programming errors and time. Flexibility is also a major benefit of the Lyee methodology since the maintenance task can be reduced to a simple modification of requirements (add, remove and/or modify words' definitions).

Consequently, Lyee systems can be viewed as collections of independent components (satement) that interact together in order to produce the desired result. The concepts of our process algebra sticks very well with this view since the Lyee-calculus considers a program as a chemical solution of molecules (processes) that interact together to reach the final goal. So, at a first glance, one can view a Lyee requirement $LR$ composed of as a set of statements $\{s_1, \ldots, s_n\}$ as a concurrent processes of the Lyee-calculus, i.e:

$$LR = s_1 \mid \ldots \mid s_n$$

### *3.2 Pallets and Scenario Function*

From the requirements in Table 3, a program can be automatically generated that computes the values of $a$ and $b$ and outputs them. This program will simply repeat the execution of these instructions until a fixed point is reached, i.e. any other iteration will not change the value of any word as shown in Fig. 3.

From the process algebra view point, the notion of "reaching of a fixed point" is naturally implemented within the semantic. In fact, the processes (molecules) interact together until a state of no possible evolution (fix point) is reached.

Let's be more precise about the structure and the content of the program that will be automatically generated by Lyee from requirements. Within the Lyee methodology, the execution

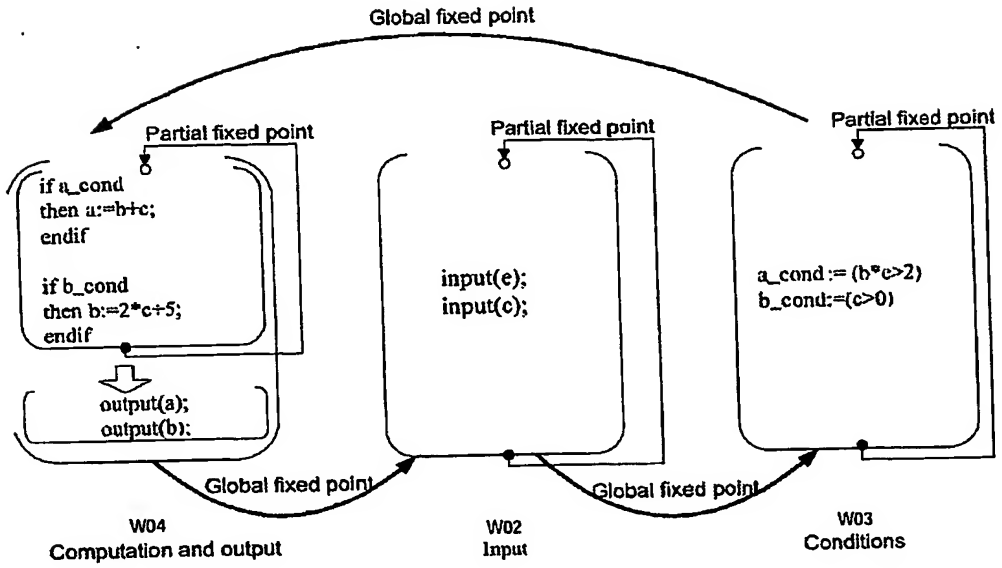



Figure 4: Lyee Pallets.

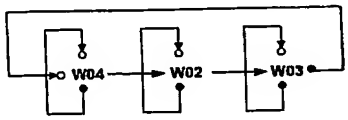



Figure 5: Scenario Function.

of a set of statements, such as the ones given in Table 3, is accomplished in a particular manner. In fact, Lyee distributes the code associated with statements over three spaces, called *Pallets* ($W_{02}$, $W_{03}$ and $W_{04}$) in the Lyee terminology, as shown in Fig. 4.

Pallet $W_{02}$ deals with the input words. Pallet $W_{03}$ computes the calculation conditions of the words and saves the results in boolean variables. For instance, the condition "b*e>2" used within the definition of the word "a" is calculated in $W_{03}$ and the true/false result is saved in another variable "a_cond". Finally, pallet $W_{04}$ deals with the calculation of the words according to their definition given within the requirements. It also outputs the values of the computed words.

Starting from pallet $W_{04}$, a Lyee program tries to compute the values of all the defined words until a fixed point is reached. Once there is no evolution in $W_{04}$ concerning the computation of the word values, control is given, by routing vector $R4C$, to pallet $W_{02}$. In its turn, this second pallet tries repeatedly to input values of words until a fixed point is reached (no others inputs are available) and then, by routing vector $R2C$, transfer the control to pallet $W_{03}$. Finally, and similar to pallet $W_{04}$, pallet $W_{03}$ tries to compute the calculation conditions of the words according to the requirements until a fixed point is reached. As shown in Fig. 5, this whole process ($W_{04} \longrightarrow W_{02} \longrightarrow W_{03}$) will repeat until a situation of overall stability is reached and the three pallet linked together are called Scenario Function.

Using a sequential langage to implement a Lyee requirement forces the author of the Lyee

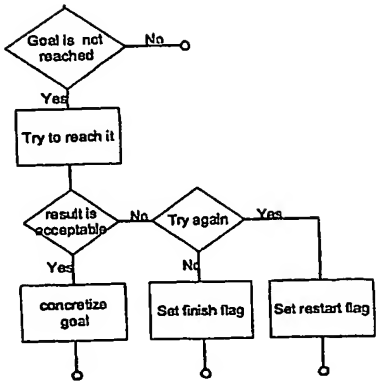



Figure 6: Predicate Vector.

methodology to explicitly specify the order in which the pallet will be executed, the order according to which vectors in the same pallet have to be executed and how to pass control from one vector to another. In other word, this sequential langage forces the introduction of some aspects (routing vectors $RC2$, $RC3$, $RC4$, etc., tense control functions $\Phi_2$, $\Phi_3$, $\Phi_4$, etc.) that do not belongs to fondamental ideas of the Lyee methodology, and that they have considerably complicated this methodology.

However, using the Lyee-calculus, we do not need any more to specify the order according to which the pallets will be executed, the order according to which the vectors of each pallet will be executed, how and when to pass control from one pallet to another, etc. All this detail will be naturally managed by our abstract machine. Consequently, a Scenario Function could be formalized as follows:

$$SF = W_{04} \mid W_{03} \mid W_{02}$$

Notice that the order according to which $W_{04}$, $W_{03}$ and $W_{02}$ is no more important and the routing vectors that connect these pallets are also eliminated.

Lyee has established a simple program with a fixed structure (called a Predicate Vector in the Lyee terminology) that makes the structure of generated code uniform and independent of the requirement content. The global program will be simple calls of predicate vectors. The structure of a predicate vector is as shown in Fig. 6.

The goal of a predicate vector changes from one pallet to another. For instance, in the pallet $W_{04}$, the first goal is to give a value to a word according to its definition. For the example shown in Fig. 4, the predicate vectors associated with the calculation of the word "a" and the word "b" are as shown in Fig. 7.

Once there is no evolution in the calculation of the words, the Lyee generated code tries to output the words which will be the next goal. The predicate vector having the goal of outputting values is called the output vector. In pallet $W_{02}$, we find two predicate vectors having as goal the associating of values with input words. Finally, in pallet $W_{03}$, the goal of the predicate vectors is to compute preconditions specified within requirements as shown in Fig. 8.

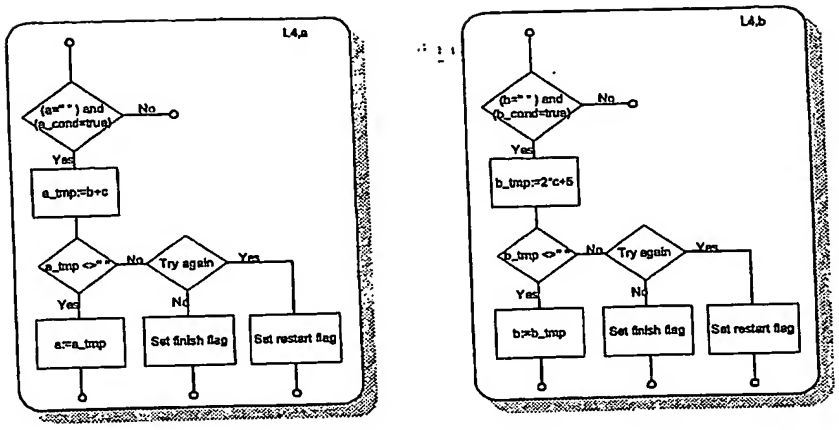



Figure 7: The Predicate Vectors of $L4, a$ and $L4, b$.

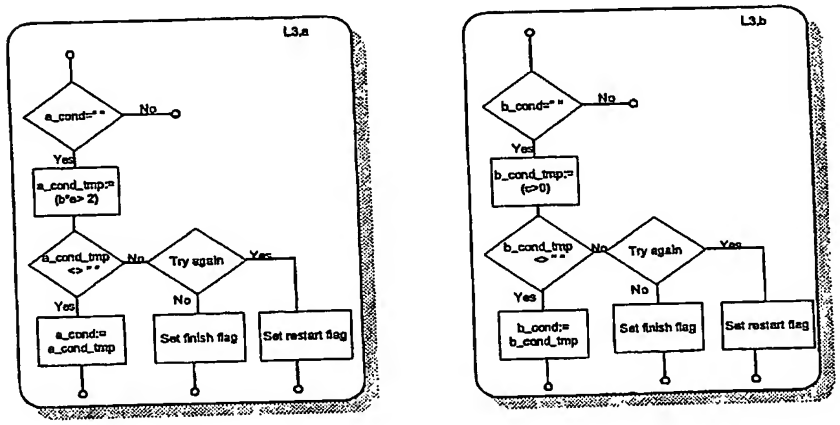



Figure 8: The Predicate Vectors of $L3, a$ and $L3, b$.

Finally, the Lyee program associated with the requirements given in Table 3 is as shown in Table 5.

Now, using the Lyee-calculus, $W_{04}$, $W_{03}$ and $W_{02}$ can be specified as follows:

$$W_{04} = S4 \mid L4\_a \mid L4\_b \mid O4$$

$$W_{03} = L3\_a \mid L3\_b$$

$$W_{02} = L2\_e \mid L2\_c \mid I2$$

The formal definition of the different vectors (L2, L3, L4, S4, etc.) will be given in the next section.

### 3.3 *Process Route Diagrams*

The Scenario Function presented in the previous section can be a complete program for a simple case of given requirements, particulary when all the input and output words belong

Table 5: Lyee Generated Program.

| Pallet | Program | Comments |
|---|---|---|
| $W_{04}$ | Call S4<br>Do<br>Call L4_a<br>Call L4_b<br>while a fixed point is not reached<br>Call O4<br>Call R4 | Initialize memory<br><br>Calculate a<br>Calculate b<br><br>Output the result<br>Go to $W_{02}$ |
| $W_{02}$ | Do<br>Call L2_e<br>Call L2_c<br>while a fixed point is not reached<br>Call I2<br>Call R2 | <br><br><br><br>Input results<br>Go to $W_{03}$ |
| $W_{03}$ | Do<br>Call L3_a<br>Call L3_b<br>while a fixed point is not reached<br>Call R3 | <br>Calculate a_cond<br>Calculate b_cond<br><br>Go to $W_{04}$ |

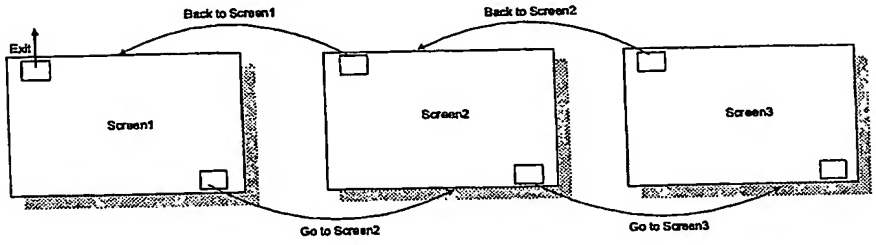



Figure 9: Screen Interactions.

to the same screen and there is no use of any database. However, if we need to input and output words that belong to databases or to different screens interconnected together, then the situation will be more complicated. For the sake of simplicity, we deal, in the sequel, only with the case when we have many screens. Suppose for instance that we have three interconnected screens, as shown in Fig. 9, allowing a user to navigate from one to another. Suppose in each one of them the user can input, compute and output some words. Therefore, in the specification, the user must specify how these screens are interconnected.

Furthermore, it is not convenient to define only one scenario function in which we compute all the words defined in all the screens. In fact, some screens may not be visited for a given execution of the program and then the computation of the value of their words will be lost. For that reason, Lyee associates with each screen its owner scenario function that will be executed only if this screen is visited. The scenario functions associated with screens are connected together showing the move from one of them to another. In the Lyee terminology, many scenario functions connected together make up a Process Route Diagram as shown in Fig. 10.

Using the Lyee-calculus, we can formalize a screen $s_k$ as a process $\Phi(s_k)$ that control the execution of the process $SF(s_k)$. Actually, the process $SF(s_k)$ will be executed until it corresponding screen is reactivates (receiving a signal on the channel $z^{s_k}$ associated to the

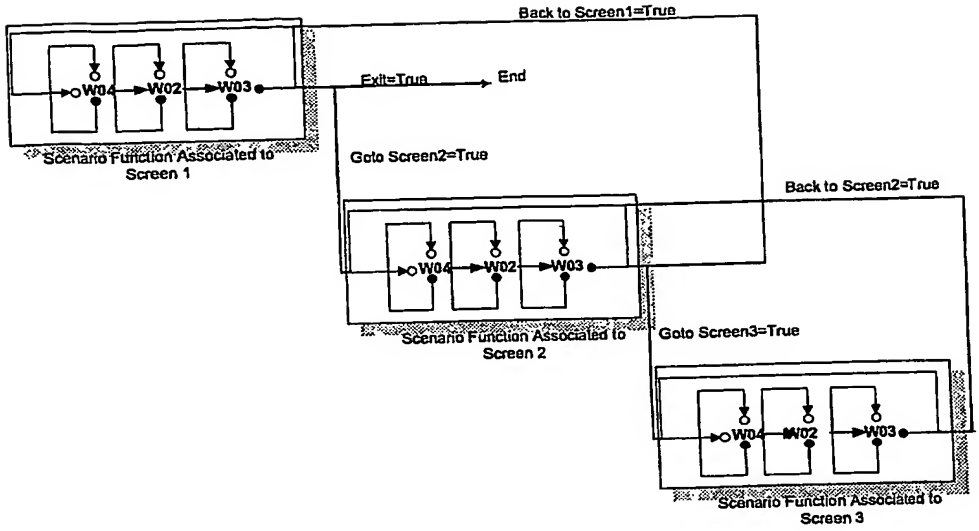



Figure 10: Processes Route Diagram.

screen $s_k$) and then will be killed to create a fresh instance of $SF(s_k)$. This is formalized as follows:

$$\Phi(s_k) = SF(s_k) \rhd [i^{s_k}?1].\Phi(s_k)$$

Notice that a concret example will be given later in this paper to show how the function $\Phi(s_k)$ works.

### *3.4 Lyee Program*

To summarize, according to the Lyee methodology, a Lyee program contains several processus route diagrams (PRD). Each of them is a set of interconnected scenario functions (SF). Each scenario function contains three interconnected pallets $W_{02}$, $W_{03}$ and $W_{04}$. Finally, pallet contains atomic vectors (signification vectors and action vectors).

Using the Lyee-calculus, a Lyee program $\mathcal{P}$ containing the screens $s_1, \ldots, s_k$, is formalized as follows:

$$\mathcal{P}(s_1, \ldots, s_k) = \Psi(s_1, \ldots, s_k)/\mathcal{L}(s_1, \ldots, s_k)$$

where $\mathcal{L}(s_1, \ldots, s_k)$ is the set of input and output channels. This restricts $\Psi(s_1, \ldots, s_k)$ to communicate with the environment only on channels in $\mathcal{L}(s_1, \ldots, s_k)$.

The function $\Psi$ is defined as follows:

$$\Psi(s_1, \ldots, s_k) = (|_{s \in \{s_1, \ldots, s_k\}} [i^s?1].\Phi(s)) \rhd [i^{s_0}?1].nil$$

This function activates the SF of a screen $s_k$ whenever a true value is received on the corresponding channel $i^{s_k}$, which formalizes a button or a menu item that a user may use to activate the corresponding screen, and kill all the processes when it receives a true value on channel $i^{s_0}$ which formalizes the Exit button (or a corresponding menu item).

Consequently, a Lyee system can be viewed as a collection of independent concurrent processes that communicate with each other to compute the desired output words. In contrast with conventional sequential view of Lyee methodology, there are no need for various action vectors and control functions, no need of working memory areas, etc. More details about $\Psi(s_1, \ldots, s_k)$ together with concret examples will be given in the sequel.

The next section gives a more detailed and complete formalization of the Lyee methodology.

## 4 Formalization of the Lyee Methodology

Let Use(e) denotes the set of words used in the expression $e$. For instance, $Use(a * b + 1) = \{a, b\}$. Also, let $\mathcal{F}(S)$ be the following function:

$$\begin{aligned} \mathcal{F}(\emptyset) &= \emptyset \\ \mathcal{F}(\{x\} \cup A) &= \{i_4^x?x\} \cup \mathcal{F}(A) \end{aligned}$$

Now let's see how to automatically generate software from simple user requirement using the Lyee-calculus. Suppose that the user requirement contains $k$ screens $\{s_1, \ldots, s_k\}$. Suppose also that each screen contains a set of statements where each one of them has the following form: $(w, e, c, InOut, type)$ where $w$ is a word, $e$ its definition, $c$ its condition, $InOut$ to specify if it is input, output both or neither input no output (the value $i$ is used for input word, $o$ for output, $io$ for both input and output and empty filed for neither input nor output) and $type$ is its type (the type $B$ is reserved for bouton). To define the program $\mathcal{P}(s_1, \ldots, s_k)$ associated with this requirement, we need to formalize the following vectors (processes):

**Signification Vectors**

| Vector | Comments |
|---|---|
| $L_4(x, e) =$<br>$[i_3^x?1].$<br>$[\mathcal{F}(Use(e))].$<br>$[\jmath_4^x!e].$<br>$nil$ | wait to receive the value true (1) on the channel $i_3^x$ (from $L_3$)<br>wait to receive the value of all the $Use$ of $e$ from the memory<br>evaluate $e$ and put the result in the memory cell of $x$<br>finish |
| $L_3(x, c) =$<br>$[\mathcal{F}(Use(c))].$<br>$[i_3^x!c].$<br>$nil$ | wait to receive the value of all the $Use$ of $c$ from the memory<br>evaluate $c$ and send the result on the channel $i_3^x$ (to $L_4$)<br>finish |
| $L_2(x) =$<br>$[i_2^x?x].$<br>$[\jmath_4^x!x].$<br>$nil$ | wait to receive the value $x$ on the channel $i_2^x$<br>(from the input vector)<br>save the value of $x$ in the memory cell of $x$<br>finish |

**Action Vectors**

| Input Vector | Comments |
|---|---|
| $I_2(x) = [d_x?x].[\imath_2^x!x].nil$ | wait to receive a value on the input channel $d_x$<br>send the value of $x$ on the channel $\imath_2^x$ (to $L_2$)<br>finish |
| **Output Vector** | **Comments** |
| $O_4(x) = [\imath_4^x?x].[d_x!x].nil$ | wait to receive the value of $x$ from the memory<br>send the value of $x$ on the output channel $d_x$<br>finish |
| **Memory (Structural Vector)** | **Comments** |
| $S_4(x) = [\jmath_4^x?y].S_4^x(y)$ | wait to receive any value (initialization) and then behave as $S_4^x(y)$ |
| $S_4^x(y) = [\jmath_4^x?z].S_4^x(z) + [\imath_4^x!y].S_4^x(y)$ | if you receive $z$ behaves as $S_4^x(z)$<br>(change the content of the cell $x$)<br>if you send $y$ behaves as $S_4^x(y)$<br>(do not change the content of the cell $x$) |
| **Routing Vector** | **Comments** |
| $R_3(b,e,s) = [d_b?clik].[\mathcal{F}(Use(e))].[\imath^s!e].nil$ | wait until the bouton $b$ be pushed<br>evaluate the condition of the bouton<br>send the result to the screen $s$ (i.e., activate $s$ if $e=1$)<br>finish |

**Pallet:** The three pallets $W_{02}$, $W_{03}$ and $W_{04}$ of a given screen $s$ are formalized as follows:

$$\begin{aligned} W_{02}(s) &= |_{(w,*,*,i,\overline{B})\in s}\ I_2(w)\ |_{(w,*,*,i,\overline{B})\in s}\ L_2(w) \\ W_{03}(s) &= |_{(w,*,c,*,\overline{B})\in s}\ L_3(w,c)\ |_{(w,e,c,i,*,B)\in s}\ R_3(w,c,e) \\ W_{04}(s) &= |_{(w,*,*,*,\overline{B})\in s}\ S_4(w)\ |_{(w,e,*,*,\overline{B})\in s} L_4(w,e))\ |_{(w,*,*,o,*,*)\in s}\ O_4(w) \end{aligned}$$

where $\overline{B}$ denotes any type except $B$ (complementary of $B$) and $*$ denotes anything.

**Scenario Function:** The scenario function, $SF(s)$, attached to the screen $s$ is formalized as follows :

$$SF(s) = W_{04}(s) \mid W_{03}(s) \mid W_{02}(s)$$

**Control Function:** The control function attached to a screen $s$, is formalized as follows:

$$\Phi(s) = SF(s) \rhd [\imath^{s_k}?1].\Phi(s)$$

The control function attached to a set of screens is formalized as follows:

$$\Psi(s_1,\ldots,s_k) = (|_{s\in\{s_1,\ldots,s_k\}}\ [\imath^s?1.]\Phi(s)) \rhd [\imath^{s_0}?1].nil$$

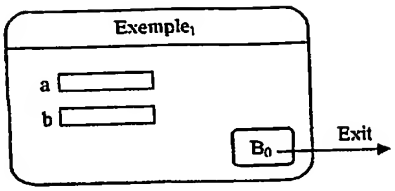



Figure 11: Case Study - One Screen.

We suppose that $s_0$ is the screen that we find when we exit the program (exit screen that does not belongs to the screens of the program itself).

**Lyee Program:** Finally, the Lyee program $\mathcal{P}(s_1, \ldots, s_k)$ associated with the requirement containing the screens $s_1, \ldots, s_k$ is as follows:

$$\mathcal{P}(s_1, \ldots, s_k) = \Psi(s_1, \ldots, s_k)/\mathcal{L}(s_1, \ldots, s_k)$$

where the set $\mathcal{L}(s_1, \ldots, s_k)$ contains all the input and output channels and it is defined as follows:

$$\mathcal{L}(s_1, \ldots, s_k) = (\bigcup_{(w,*,*,i/o,*) \in s_k} \{d_w\}) \cup \{i^{s_1}\}$$

The meaning of $i/o$ is that this field has to contains the value $i$, $o$ or $io$. We suppose also that $s_1$ is the first screen that appears when the user runs this program.

## 5 Case Study

In this section, we give a concrete example and describe step by step how a Lyee program progress in order to compute words within our Lyee-calculus.

The example that we present contains only one screen (another example with two screen is given in the appendix). As shown by the requirement given in Table 6, a user can input a word $a$, waits for the value of $b$ and then exit the screen by pushing the bouton $B_0$. The Fig. 11 illustrates such a screen.

| Word | Definition | Condition | IO | Type |
|---|---|---|---|---|
| a | | | i | real |
| b | 2*a | a>0 | o | real |
| $B_0$ | $s_0$ | i*Click* | | B |

Table 6: Lyee Requirements : Screen I.

This screen $s_1$ is composed from three statements:

$$s_1 = \{(a, , , i, real), (b, 2 * a, a > 0, o, real), (B_0, s_0, Click, B)\}$$

According to the definition of a generic Lyee program given in the previous section, the Lyee program associated to the requirement given by Table 6 is as follows:

$$SF(s_1) = W_{04}(s_1) \mid W_{03}(s_1) \mid W_{02}(s_1)$$

$$\Phi(s_1) = SF(s_1) \rhd [\imath^{s_1}?1].\Phi(s_1)$$
$$\Psi(s_1) = ([\imath^{s_1}?1].\Phi(s_1)) \rhd [\imath^{s_0}?1].nil$$
$$\mathcal{L}(s_1) = \{d_a, d_b, d_{B_0}, \imath^{s_1}\}$$

$$\mathcal{P}(s_1) = \Psi(s_1)/\mathcal{L}(s_1)$$

where $W_{02}(s_1)$, $W_{03}(s_1)$ and $W_{04}(s_1)$ are as shown hereafter:

| $W_{02}(s_1) = L_2(a) \mid I_2(a)$ | |
|---|---|
| $L_2(a) = [\imath_2^a?a].$<br>$[\jmath_4^a!a].$<br>$nil$ | $I_2(a) = [d_a?a].$<br>$[\imath_2^a!a].$<br>$nil$ |

| $W_{03}(s_1) = L_3(b, a > 0) \mid R_3(B_0, click, s_0)$ | |
|---|---|
| $L_3(b, a > 0) = [\imath_4^a?a].$<br>$[\imath_3^b!(a > 0)].$<br>$nil$ | $R_3(B_0, click, s_0) = [d_{B_0}?click].$<br>$[\imath^{s_0}!1].$<br>$nil$ |

| $W_{04}(s_1) = S_4(a) \mid S_4(b) \mid L_4(b, 2 * a) \mid O_4(b)$ | | |
|---|---|---|
| $S_4(a) = [\jmath_4^a?a].$<br>$S_4^a(a)$<br>$S_4(b) = [\jmath_4^b?b].$<br>$S_4^b(b)$ | $L_4(b, 2 * a) = [\imath_3^b?1].$<br>$[\imath_4^a?a].$<br>$[\jmath_4^b!(2 * a)].$<br>$nil$ | $O_4(b) = [\imath_4^b?b].$<br>$[d_b!b].$<br>$nil$ |

Suppose now, that the end-user (environment) of this program wants to perform the following sequence of actions : runs the program (activates the screen $s_1$), gives the value 7 to the word $a$, waits for the value of the word $b$ and then exits the program by pushing the bouton $B_0$. This comportement can be captured by the following process $\mathcal{E}$:

| Process | Comments |
|---|---|
| $\mathcal{E} = [\imath^{s_1}!1].\mathcal{E}_1$ | activate the program (screen $s_1$) |
| $\mathcal{E}_1 = [d_a!7].\mathcal{E}_2$ | give the value 7 to the word $a$ |
| $\mathcal{E}_2 = [d_b?x].\mathcal{E}_3$ | get the value of the word $b$ |
| $\mathcal{E}_3 = [d_{B_0}!click].nil$ | clicke the bouton $B_0$ to exit |

Let's see now how the program interact with this environment to reach the desired goal. In other words, we want to have a trace of the program progress when executed in parallel with the program $\mathcal{E}$ that implements the end-used actions, i.e., we look for the behaviors of $\mathcal{P}(s_1) \mid \mathcal{E}$. The steps of this trace is as shown hereafter :

$$
\begin{aligned}
& \mathcal{P}(s_1) \mid \mathcal{E} \\
= & \quad \langle \text{ By definition of } \mathcal{P} \text{ and } \mathcal{E} \rangle \\
& \Psi(s_1)/\mathcal{L}(s_1) \mid [i^{s_1}!1].\mathcal{E}_1 \\
= & \quad \langle \text{ By definition of } \Psi \rangle \\
& \big(([\underline{i^{s_1}?1}].\Phi(s_1)) \rhd [i^{s_0}?1].nil\big)/\mathcal{L}(s_1) \mid \underline{[i^{s_1}!1]}.\mathcal{E}_1 \\
\xrightarrow{\tau} & \quad \langle \text{ Synchronization} \\
& \quad \textit{– End-user activates the program by running the screen } s_1 \rangle \\
& \big(\Phi(s_1) \rhd [i^{s_0}?1].nil\big)/\mathcal{L}(s_1) \mid \mathcal{E}_1 \\
= & \quad \langle \text{ By definition of } \Phi \text{ and } \mathcal{E}_1 \rangle \\
& \big((SF(s_1) \rhd [i^{s_1}?1].\Phi(s_1)) \rhd [i^{s_0}?1].nil\big)/\mathcal{L}(s_1) \mid [d_a!7].\mathcal{E}_2 \\
= & \quad \langle \text{ By definition of } SF \rangle \\
& \big(((W_{04}(s_1) \mid W_{02}(s_1) \mid W_{03}(s_1)) \rhd [i^{s_1}?1].\Phi(s_1)) \rhd [i^{s_0}?1].nil\big)/\mathcal{L}(s_1) \mid [d_a!7].\mathcal{E}_2 \\
= & \quad \langle \text{ By definition of } W_{02} \text{ and consequently } I_2 \rangle \\
& \big(((W_{04}(s_1) \mid L_2(a) \mid ([\underline{d_a?a}].[i_2^a!a].nil) \mid W_{03}(s_1)) \rhd \dots) \rhd \dots\big)/\mathcal{L}(s_1) \mid \underline{[d_a!7]}.\mathcal{E}_2 \\
\xrightarrow{\tau} & \quad \langle \text{ Synchronization} \\
& \quad \textit{– Through } I_2 \textit{ end-user gives the value 7 to the word } a \rangle \\
& \big(((W_{04}(s_1) \mid L_2(a) \mid ([i_2^a!7].nil) \mid W_{03}(s_1)) \rhd \dots) \rhd \dots\big)/\mathcal{L}(s_1) \mid \mathcal{E}_2 \\
= & \quad \langle \text{ By definition of } L_2 \rangle \\
& \big(((W_{04}(s_1) \mid ([\underline{i_2^a?a}].[\jmath_4^a!a].nil) \mid (\underline{[i_2^a!7]}.nil) \mid W_{03}(s_1)) \rhd \dots) \rhd \dots\big)/\mathcal{L}(s_1) \mid \mathcal{E}_2 \\
\xrightarrow{\tau} & \quad \langle \text{ Synchronization} \\
& \quad \textit{– The value of } a \textit{ is communicated to } L_2(a) \rangle \\
& \big(((W_{04}(s_1) \mid ([\jmath_4^a!7].nil) \mid W_{03}(s_1)) \rhd \dots) \rhd \dots\big)/\mathcal{L}(s_1) \mid \mathcal{E}_2 \\
= & \quad \langle \text{ By definition of } W_{04} \rangle \\
& \big((S_4(a) \mid S_4(b) \mid L_4(b, 2*a) \mid O_4(b) \mid ([\jmath_4^a!7].nil) \mid W_{03}(s_1)) \rhd \dots\big)/\mathcal{L}(s_1) \mid \mathcal{E}_2 \\
= & \quad \langle \text{ By definition of } S_4(a) \rangle \\
& \big(((\underline{[\jmath_4^a?a]}.S_4^a(a)) \mid S_4(b) \mid L_4(b, 2*a) \mid O_4(b) \mid (\underline{[\jmath_4^a!7]}.nil) \mid W_{03}(s_1)) \rhd \dots \\
\xrightarrow{\tau} & \quad \langle \text{ Synchronization} \\
& \quad \textit{– The value of } a \textit{ is stored in the memory} \rangle \\
& \big((S_4^a(7) \mid S_4(b) \mid L_4(b, 2*a) \mid O_4(b) \mid W_{03}(s_1)) \rhd \dots \\
= & \quad \langle \text{ By definition of } S_4^a(7) \text{ and } W_{03} \rangle \\
& \big(((([\jmath_4^a?z].S_4^a(z) + [i_4^a!7].S_4^a(7)) \mid \dots \mid (L_3(b, a>0) \mid R_3(B_0, click, s_0))) \rhd \dots \\
= & \quad \langle \text{ By definition of } L_3 \rangle \\
& \big(((([\jmath_4^a?z].S_4^a(z) + \underline{[i_4^a!7]}.S_4^a(7)) \mid \dots \mid (([\underline{i_4^a?a}].[i_3^b!(a>0)].nil) \mid \dots \\
\xrightarrow{\tau} & \quad \langle \text{ Synchronization} \\
& \quad \textit{– The value of } a \textit{ is communicated to } L_3(b) \textit{ to compute the } b \textit{ condition} \rangle \\
& \big((S_4^a(7) \mid S_4(b) \mid L_4(b, 2*a) \mid O_4(b) \mid ([i_3^b!(7>0)].nil) \mid \dots \\
= & \quad \langle \text{ By definition of } L_4 \text{ and evaluation of } (7>0) \rangle \\
& \big((S_4^a(7) \mid S_4(b) \mid (\underline{[i_3^b?1]}.[i_4^a?a].[\jmath_4^b!(2*a)].nil) \mid O_4(b) \mid (\underline{[i_3^b!1]}.nil) \mid \dots
\end{aligned}
$$

$\xrightarrow{\tau}$ ⟨ Synchronization
– *The b condition is evaluated and communicated to* $L_4(b)$ *to activate the computation of b* ⟩
$( (S_4^a(7) \mid S_4(b) \mid ([i_4^a?a].[j_4^b!(2*a)].nil) \mid O_4(b) \mid \ldots$

$=$ ⟨ By definition of $S_4^a(7)$ ⟩
$( (([j_4^a?z].S_4^a(z) + \underline{[i_4^a!7]}.S_4^a(7)) \mid S_4(b) \mid (\underline{[i_4^a?a]}.[j_4^b!(2*a)].nil) \mid O_4(b) \mid \ldots$

$\xrightarrow{\tau}$ ⟨ Synchronization
– *The value of a is communicated to* $L_4$ ⟩
$( (S_4^a(7) \mid S_4(b) \mid ([j_4^b!(2*7)].nil) \mid O_4(b) \mid R_3(B_0, click, s_0)) \rhd \ldots$

$=$ ⟨ By definition of $S_4(b)$ and evaluation of $(2*7)$ ⟩
$( (S_4^a(7) \mid (\underline{[j_4^b?b]}.S_4^b(b)) \mid (\underline{[j_4^b!(14)]}.nil) \mid O_4(b) \mid R_3(B_0, click, s_0)) \rhd \ldots$

$\xrightarrow{\tau}$ ⟨ Synchronization
– *The value of b (14) is stored in the memory* ⟩
$( (S_4^a(7) \mid S_4^b(14) \mid O_4(b) \mid R_3(B_0, click, s_0)) \rhd \ldots$

$=$ ⟨ By definition of $S_4^b(14)$ and $O_4(b)$ ⟩
$( (S_4^a(7) \mid ([j_4^b?z].S_4^b(z) + \underline{[i_4^b!14]}.S_4^b(14)) \mid (\underline{[i_4^b?b]}.[d_b!b].nil) \mid R_3(B_0, click, s_0)) \rhd \ldots$

$\xrightarrow{\tau}$ ⟨ Synchronization
– *The b value is communicated to* $O_4$ ⟩
$( (S_4^a(7) \mid S_4^b(14) \mid ([d_b!14].nil) \mid R_3(B_0, click, s_0)) \rhd \ldots )/\mathcal{L}(s_1) \mid \mathcal{E}_2$

$=$ ⟨ By definition of $\mathcal{E}_2$ ⟩
$( (S_4^a(7) \mid S_4^b(14) \mid (\underline{[d_b!14]}.nil) \mid R_3(B_0, click, s_0)) \rhd \ldots )/\mathcal{L}(s_1) \mid \underline{[d_b?x]}.\mathcal{E}_3$

$\xrightarrow{\tau}$ ⟨ Synchronization
– *The b value, stored in* $O_4$*, is communicated to the environment* ⟩
$( (S_4^a(7) \mid S_4^b(14) \mid R_3(B_0, click, s_0)) \rhd \ldots )/\mathcal{L}(s_1) \mid \mathcal{E}_3$

$=$ ⟨ By definition of $R_3$ and $\mathcal{E}_3$ ⟩
$( (S_4^a(7) \mid S_4^b(14) \mid (\underline{[d_{B_0}?click]}.[i^{s_0}!1].nil)) \rhd \ldots )/\mathcal{L}(s_1) \mid \underline{[d_{B_0}!click]}.nil$

$\xrightarrow{\tau}$ ⟨ Synchronization
–*The end-user push button* $B_0$ *to exit* ⟩
$( (S_4^a(7) \mid S_4^b(14) \mid (\underline{[i^{s_0}!1].nil})) \rhd [i^{s_1}?1].\Phi(s_1)) \rhd \underline{[i^{s_0}?1].nil} )/\mathcal{L}(s_1)$

$\xrightarrow{\tau}$ ⟨ Synchronization
–*The system ends its execution* ⟩

$nil$

To summarize, here are the main steps of the program progress:

1. End-user activates the program by running the screen $s_1$.
   $( (\underline{[i^{s_1}?1]}.\Phi(s_1)) \rhd [i^{s_0}?1].nil )/\mathcal{L}(s_1) \mid \underline{[i^{s_1}!1]}.\mathcal{E}_1$
2. Through $I2(a)$, end-user gives the value 7 to the word $a$.
   $( ( (W_{04}(s_1) \mid L_2(a) \mid (\underline{[d_a?a]}.[i_2^a!a].nil) \mid W_{03}(s_1)) \rhd \ldots ) \rhd \ldots )/\mathcal{L}(s_1) \mid \underline{[d_a!7]}.\mathcal{E}_2$
3. The value of $a$ is communicated to $L_2(a)$.
   $( ( (W_{04}(s_1) \mid (\underline{[i_2^a?a]}.[j_4^a!a].nil) \mid (\underline{[i_2^a!7]}.nil) \mid W_{03}(s_1)) \rhd \ldots ) \rhd \ldots )/\mathcal{L}(s_1) \mid \mathcal{E}_2$

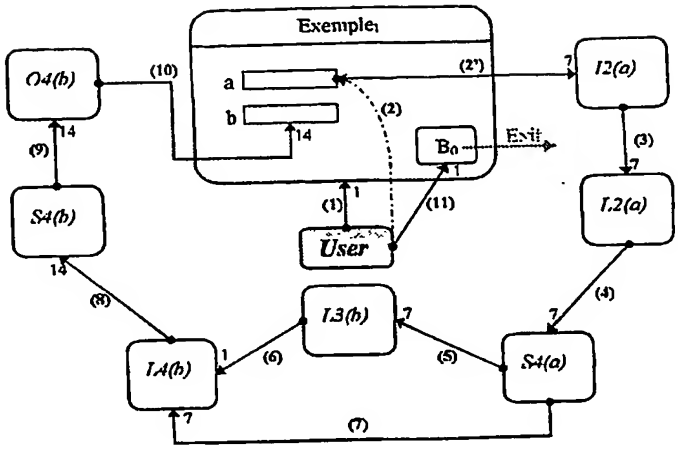



Figure 12: Interaction between processes.

4. The value of $a$ is stored in the memory, i.e. in $S4(a)$.
   $(((\underline{[\jmath_4^a?a]}.S_4^a(a)) \mid S_4(b) \mid L_4(b, 2*a) \mid O_4(b) \mid (\underline{[\jmath_4^a!7]}.nil) \mid W_{03}(s_1)) \rhd \ldots$

5. The value of $a$ is communicated to $L_3(b)$ to compute the $b$ condition.
   $(((\underline{[\jmath_4^a?z]}.S_4^a(z) + \underline{[\imath_4^a!7]}.S_4^a(7)) \mid \ldots \mid ((\underline{[\imath_4^a?a]}.[\imath_3^b!(a > 0)].nil) \mid \ldots$

6. The $b$ condition is evaluated and communicated to $L_4(b)$.
   $((S_4^a(7) \mid S_4(b) \mid (\underline{[\imath_3^b?1]}.[\imath_4^a?a].[\jmath_4^b!(2*a)].nil) \mid O_4(b) \mid (\underline{[\imath_3^b!1]}.nil) \mid \ldots$

7. The value of $a$ is communicated, from memory, to $L_4(b)$.
   $(((\underline{[\jmath_4^a?z]}.S_4^a(z) + \underline{[\imath_4^a!7]}.S_4^a(7)) \mid S_4(b) \mid (\underline{[\imath_4^a?a]}.[\jmath_4^b!(2*a)].nil) \mid O_4(b) \mid \ldots$

8. The value of $b$ is stored in the memory, i.e. in $S4(b)$.
   $((S_4^a(7) \mid (\underline{[\jmath_4^b?b]}.S_4^b(b)) \mid (\underline{[\jmath_4^b!(14)]}.nil) \mid O_4(b) \mid R_3(B_0, click, s_0)) \rhd \ldots$

9. The $b$ value is communicated to the output vector $O4(b)$.
   $((S_4^a(7) \mid (\underline{[\jmath_4^b?z]}.S_4^b(z) + \underline{[\imath_4^b!14]}.S_4^b(14)) \mid (\underline{[\imath_4^b?b]}.[d_b!b].nil) \mid R_3(B_0, click, s_0)) \rhd \ldots$

10. The $b$ value, stored in $O_4$, is communicated to the environment.
    $((S_4^a(7) \mid S_4^b(14) \mid (\underline{[d_b!14]}.nil) \mid R_3(B_0, click, s_0)) \rhd \ldots)/\mathcal{L}(s_1) \mid \underline{[d_b?x]}.\mathcal{E}_3$

11. The end-user push button $B_0$ to exit.
    $((S_4^a(7) \mid S_4^b(14) \mid (\underline{[d_{B_0}?click]}.[\imath^{s_0}!1].nil)) \rhd \ldots)/\mathcal{L}(s_1) \mid \underline{[d_{B_0}!click]}.nil$

The Fig. 12 illustrates each of these steps.
Furtheremore, each transition could be validated by a proof. For instance, the transition:
$(\mathcal{P}(s_1) \mid \mathcal{E}) \xrightarrow{\tau} (\mathcal{P}_1(s_1) \mid \mathcal{E}_1)$
where:

$$\mathcal{P}_1(s_1) = (\Phi(s_1) \rhd [\imath^{s_0}?1].nil)/\mathcal{L}(s_1)$$

is justified by the following proof:

$$R_? : \frac{1 \in [\![1]\!]}{[?^{s_1}?1].\Phi(s_1) \xrightarrow{?^{s_1}?1} \Phi(s_1)} \qquad R^l_{\triangleright} : \frac{\Psi(s_1) \xrightarrow{?^{s_1}?1} \Phi(s_1) \triangleright [?^{s_0}?1].nil}{\mathcal{P}(s_1) \xrightarrow{?^{s_1}?1} \mathcal{P}_1(s_1)} \qquad R_! : \frac{1 \in [\![1]\!]}{\mathcal{E} \xrightarrow{!^{s_1}!1} \mathcal{E}_1}$$

$$R^{\tau}_{|} : \frac{}{(\mathcal{P}(s_1) \mid \mathcal{E}) \xrightarrow{\tau} (\mathcal{P}_1(s_1) \mid \mathcal{E}_1)}$$

## 6 Conclusion

In this paper, we have first defined a formal process algebra, called Lyee-calculus, that easily and naturally supports the basic concepts of the Lyee methodology. In fact, this calculus can be seen as an abstract machine which is more suitable to support the Lyee methodology concepts than the Von Newman one. This machine considers a program as a set of molecules that interact together to produce a final result. Secondly, we have shown how this calculus can formalize and simplify both the lyee vectors together with the whole software generation process. Actually, synchronization vector, most of the routing vectors, command vector, etc., are no longer needed. We have also formalized the whole process of the automatic generation of Lyee software generation. This formalization offers to this methodology a formal semantics that allows one to clearly understand the concepts behind this methodology. Besides, this formal description will be an inevitable start point for many interesting analysis on diverse aspects of this methodology. For instance, to optimize the Lyee generated software or the software generation processes, we need a formal proof that the optimized program is *equivalent* to its original version. Formal equivalence checking or more generally model checking can be elegantly achieved when using process algebra.

As future work, we want to more investigate the semantics of Lyee methodology in order to make it more simple and to generate, from requirements, more reliable and optimized code. For the optimization end, we intend to define a congruence relation between processes of Lyee-calculus and then using it to prove the correctness of all the eventual optimizations.

## References


[1] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[2] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96(1):217–248, 1992.

[3] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science. Prentice Hall, 1985.

[4] M. Mejri, B. Ktari, and M. Erhioui. Static analysis on lyee-oriented software. In Hamido Fujita and Paul Johannesson, editors, *New Trends in Software Methodologies, Tools and Techniques*, pages 375–394. IOS Press, 2002. Proceedings of 1st International Workshop on Lyee Methodology, Paris.

[5] R. Milner. A calculus of communicating systems. Lecture Notes in Computer Science 92, Berlin, 1980. Springer-Verlag.

[6] F. Negoro. Principle of Lyee software. *2000 International Conference on Information Society in 21st Century (IS2000)*, pages 121–189, November 2000.

[7] F. Negoro. *Introduction to Lyee*. The Institute of Computer Based Software Methodology and Technology, Tokyo, Japan, 2001.

[8] F. Negoro and I. Hamid. A proposal for intention engineering. *5th East-European Conference Advances in Databases and Information System (ADBIS'2001)*, September 2000.

[9] F. Negoro and I. Hamid. A proposal for intention engineering. *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2001)*, 2001.

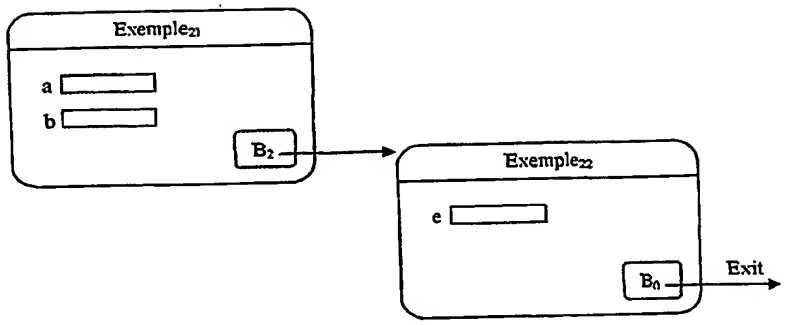



Figure 13: Case Study - Two Screen.

## Appendix

*Case Study - Two screen*

The idea behind this example is to show the interaction between many screen.

| Word | Definition | Condition | IO | ... |
|---|---|---|---|---|
| a | | | IS | ... |
| b | 2*a | a>0 | | ... |
| $B_2$ | $s_2$ | *click* | | ... |

Table 7: Lyee Requirements : Screen I.

| Word | Definition | Condition | IO | ... |
|---|---|---|---|---|
| e | 1+b | b>0 | OS | ... |
| $B_0$ | $s_0$ | *click* | | ... |

Table 8: Lyee Requirements : Screen II.

According to the definition of a generic Lyee program given in the previous section, the Lyee program associated to the requirement given by Tables 7 and 8 is as follows:

$$SF(s_1) = W_{04}(s_1) \mid W_{03}(s_1) \mid W_{02}(s_1)$$
$$SF(s_2) = W_{04}(s_1) \mid W_{03}(s_1)$$

Notice that there is no $W_{02}$ for th3 screen $s_2$ since it does not contain input words other than boutons.

$$\Phi(s_1) = SF(s_1) \rhd [i^{s_1}?1].\Phi(s_1)$$
$$\Phi(s_2) = SF(s_2) \rhd [i^{s_2}?1].\Phi(s_2)$$
$$\Psi(s_1, s_2) = ([i^{s_1}?1].\Phi(s_1) \mid [i^{s_2}?1].\Phi(s_2)) \rhd [i^{s_0}?1].nil$$

$$\mathcal{L}(s_1, s_2) = \{d_a, d_b, d_e, d_{B_0}, d_{B_2}\}$$

$$\mathcal{P}(s_1, s_2) = \Psi(s_1, s_2)/\mathcal{L}(s_1, s_2)$$

where:

| $W_{02}(s_1) = L_2(a) \mid I_2(a)$ | |
|---|---|
| $L_2(a) = [\imath_2^a?a].$<br>$[\imath_4^a!a].$<br>$nil$ | $I_2(a) = [d_a?a].$<br>$[\imath_2^a!a].$<br>$nil$ |

| $W_{03}(s_1) = L_3(b, a > 0) \mid R_3(B_2, click, s_2)$ | |
|---|---|
| $L_3(b, a > 0) = [\imath_4^a?a].$<br>$[\imath_3^b!(a > 0)].$<br>$nil$ | $R_3(B_2, click, s_2) = [d_{B_2}?click].$<br>$[\imath^{s_2}!1].$<br>$nil$ |
| $W_{03}(s_2) = L_3(e, b > 0) \mid R_3(B_0, click, s_0)$ | |
| $L_3(e, b > 0) = [\imath_4^b?b].$<br>$[\imath_3^e!(b > 0)].$<br>$nil$ | $R_3(B_0, click, s_0) = [d_{B_0}?click].$<br>$[\imath^{s_0}!1].$<br>$nil$ |

| $W_{04}(s_1) = S_4(a) \mid S_4(b) \mid L_4(b, 2 * a) \mid O_4(b)$ | | |
|---|---|---|
| $S_4(a) = [\jmath_4^a?a].$<br>$S_4^a(y)$<br>$S_4(b) = [\jmath_4^b?b].$<br>$S_4^b(y)$ | $L_4(b, 2 * a) = [\imath_3^b!1].$<br>$[\imath_4^a?a].$<br>$[\jmath_4^b!(2 * a)].$<br>$nil$ | $O_4(b) = [\imath_4^b?b].$<br>$[d_b!b].$<br>$nil$ |
| $W_{04}(s_2) = S_4(e) \mid L_4(e, 1 + b) \mid O_4(e)$ | | |
| $S_4(e) = [\jmath_4^e?e].$<br>$S_4^e(y)$ | $L_4(e, 1 + b) = [\imath_3^e!1].$<br>$[\imath_4^b?b].$<br>$[\jmath_4^e!(1 + b)].$<br>$nil$ | $O_4(e) = [\imath_4^e?e].$<br>$[d_e!e].$<br>$nil$ |

$$s_1 = \{(a, , , i, s), (b, 2 * a, a > 0, o, s)\}$$

Suppose now, that the end-user (environment) of this program wants to perform the following sequence of actions : runs the program (activates the screen $s_1$), gives the value 7 to the word $a$, waits for the value of the word $b$, pushes the bouton $B_2$ to go to the screen $s_2$, waits for the value $e$ and then exits the program by pushing the bouton $B_0$. This comportement can be captured by the following process $\mathcal{E}$:

| End-user Action | Comments |
|---|---|
| $\mathcal{E} = [\imath^{s_1}!1].\mathcal{E}_1$ | activate the program |
| $\mathcal{E}_1 = [d_a!7].\mathcal{E}_2$ | give the value 7 to word $a$ |
| $\mathcal{E}_2 = [d_b?b].\mathcal{E}_3$ | wait for the value of the word $b$ |
| $\mathcal{E}_3 = [\imath^{B_2}!click].\mathcal{E}_4$ | pushes the botom $B_2$ to go screen 2 |
| $\mathcal{E}_4 = [d_e?e].\mathcal{E}_5$ | wait for the value of the word $e$ |
| $\mathcal{E}_5 = [\imath^{B_0}!click].nil$ | push the botom $B_0$ to exit |

1. End-user activates the program by running the screen $s_1$:
   $\mathcal{P}(s_1, s_2) \mid \mathcal{E} \xrightarrow{\tau} \mathcal{P}_1(s_1, s_2) \mid \mathcal{E}_1$
   where:

$$\mathcal{P}_1(s_1, s_2) = (\Phi(s_1) \mid [\imath^{s_2}?1].\Phi(s_2) \rhd [\imath^{s_0}?1].nil)/\mathcal{L}(s_1, s_2)$$

2. The end-user gives the value 7 to the word $a$.
   $\mathcal{P}_1(s_1, s_2) \mid \mathcal{E}_1 \xrightarrow{\tau} \mathcal{P}_2(s_1, s_2) \mid \mathcal{E}_2$
   where:

$$\begin{aligned}\mathcal{P}_2(s_1, s_2) \;=\; & ((((W_{04}(s_1) \mid [\imath^a_2!7].nil \mid L_2(a) \mid W_{03}(s_1)) \rhd [\imath^{s_1}?1].\Phi(s_1) \mid [\imath^{s_2}?1].\Phi(s_2)) \\ & \rhd [\imath^{s_0}?1].nil)/\mathcal{L}(s_1, s_2)\end{aligned}$$

3. The value of $a$ is stored in the memory: This will be done in two steps. First the value of $a$ is communicated to $L_2(a)$ and second $L_2(a)$ will communicate this value to the memory.
   The first step is as follows:

$$\mathcal{P}_2(s_1, s_2) \mid \mathcal{E}_2 \xrightarrow{\tau} \mathcal{P}_3(s_1, s_2) \mid \mathcal{E}_2$$

   where:

$$\begin{aligned}\mathcal{P}_3(s_1, s_2) \;=\; & (((W_{04}(s_1) \mid [\jmath^a_4!7].nil \mid W_{03}(s_1)) \rhd [\imath^{s_1}?1].\Phi(s_1) \mid [\imath^{s_2}?1].\Phi(s_2)) \\ & \rhd [\imath^{s_0}?1].nil)/\mathcal{L}(s_1, s_2)\end{aligned}$$

   In the second, the value of $a$ will be stored in the memory of $W_{04}$:

$$\mathcal{P}_3(s_1, s_2) \mid \mathcal{E}_2 \xrightarrow{\tau} \mathcal{P}_4(s_1, s_2) \mid \mathcal{E}_2$$

   where:

$$\begin{aligned}\mathcal{P}_4(s_1, s_2) \;=\; & (((S^a_4(7) \mid S_4(b) \mid L_4(b, 2 * a) \mid O_4(b) \mid W_{03}(s_1)) \rhd [\imath^{s_1}?1].\Phi(s_1) \mid [\imath^{s_2}?1].\Phi(s_2)) \\ & \rhd [\imath^{s_0}?1].nil)/\mathcal{L}(s_1, s_2)\end{aligned}$$

4. The valu of $a$ is communicated to $L_3(b)$ to compute the $b$ condition:

$$\mathcal{P}_4(s_1, s_2) \mid \mathcal{E}_2 \xrightarrow{\tau} \mathcal{P}_5(s_1, s_2) \mid \mathcal{E}_2$$

   where:

$$\begin{aligned}\mathcal{P}_5(s_1, s_2) \;=\; & (((S^a_4(7) \mid S_4(b) \mid L_4(b, 2 * a) \mid O_4(b) \mid [\imath^b_3!(7 > 0)].nil \mid R_3(B_2, s_2, click)) \\ & \rhd [\imath^{s_1}?1].\Phi(s_1) \mid [\imath^{s_2}?1].\Phi(s_2)) \rhd [\imath^{s_0}?1].nil)/\mathcal{L}(s_1, s_2)\end{aligned}$$

5. The $b$ condition is evaluated and communicated to $L_4(b)$ to activate the computation of $b$:

$$\mathcal{P}_5(s_1, s_2) \mid \mathcal{E}_2 \xrightarrow{\tau} \mathcal{P}_6(s_1, s_2) \mid \mathcal{E}_2$$

where:

$$\begin{aligned}\mathcal{P}_6(s_1, s_2) \;=\; & (((S_4^a(7) \mid S_4(b) \mid [\jmath_4^a!(2 * a)].nil \mid O_4(b) \mid R_3(B_2, s_2, click)) \\ & \rhd [\imath^{s_1}?1].\Phi(s_1) \mid [\imath^{s_2}?1].\Phi(s_2)) \rhd [\imath^{s_0}?1].nil)/\mathcal{L}(s_1, s_2)\end{aligned}$$

6. The $b$ value is computed and communicated to $S_4(b)$, the memory area of $b$:

$$\mathcal{P}_6(s_1, s_2) \mid \mathcal{E}_2 \xrightarrow{\tau} \mathcal{P}_7(s_1, s_2) \mid \mathcal{E}_2$$

where:

$$\begin{aligned}\mathcal{P}_7(s_1, s_2) \;=\; & (((S_4^a(7) \mid S_4^b(14) \mid O_4(b)) \rhd [\imath^{s_1}?1].nil \mid R_3(B_2, s_2, click)) \\ & \rhd [\imath^{s_1}?1].\Phi(s_1) \mid [\imath^{s_2}?1].\Phi(s_2)) \rhd [\imath^{s_0}?1].nil)/\mathcal{L}(s_1, s_2)\end{aligned}$$

7. The $b$ value is outputted, i.e., communicated to the environment (end-user):

$$\mathcal{P}_7(s_1, s_2) \mid \mathcal{E}_2 \xrightarrow{\tau} \mathcal{P}_8(s_1, s_2) \mid \mathcal{E}_3$$

where:

$$\begin{aligned}\mathcal{P}_8(s_1, s_2) \;=\; & (((S_4^a(7) \mid S_4^b(14) \mid R_3(B_2, s_2, click)) \rhd [\imath^{s_1}?1].\Phi(s_1) \mid [\imath^{s_2}?1].\Phi(s_2)) \\ & \rhd [\imath^{s_0}?1].nil)/\mathcal{L}(s_1, s_2)\end{aligned}$$

8. The end-user push $B_2$ to go to screen 2:

$$\mathcal{P}_8(s_1, s_2) \mid \mathcal{E}_3 \xrightarrow{\tau} \mathcal{P}_9(s_1, s_2) \mid \mathcal{E}_4$$

where:

$$\mathcal{P}_9(s_1, s_2) = (((S_4^a(7) \mid S_4^b(14)) \rhd [\imath^{s_1}?1].\Phi(s_1) \mid \Phi(s_2)) \rhd [\imath^{s_0}?1].nil)/\mathcal{L}(s_1, s_2)$$

9. Get the value of $b$ from the memory to compute the condition of $e$ :

$$\mathcal{P}_9(s_1, s_2) \mid \mathcal{E}_3 \xrightarrow{\tau} \mathcal{P}_{10}(s_1, s_2) \mid \mathcal{E}_4$$

where:

$$\begin{aligned}\mathcal{P}_{10}(s_1, s_2) \;=\; & (((S_4^a(7) \mid S_4^b(14)) \rhd [\imath^{s_1}?1].\Phi(s_1) \mid (W_{04}(s_2) \mid [\imath_3^e(14 > 0)].nil \mid \\ & R_3(B_0, s_0, click)) \rhd [\imath^{s_1}?1].\Phi(s_1)) \rhd [\imath^{s_0}?1].nil)/\mathcal{L}(s_1, s_2)\end{aligned}$$

10. Compute the value of $e$:

$$\mathcal{P}_{10}(s_1, s_2) \mid \mathcal{E}_3 \xrightarrow{\tau} \mathcal{P}_{11}(s_1, s_2) \mid \mathcal{E}_4$$

where:

$$\begin{aligned}\mathcal{P}_{11}(s_1, s_2) \;=\; & (((S_4^a(7) \mid S_4^b(14)) \rhd [i^{s_1}?1].\Phi(s_1) \mid (S_4(e) \mid [J_4^e(1+14)].nil \mid O_4(e) \mid \\ & R_3(B_0, s_0, click)) \rhd [i^{s_1}?1].\Phi(s_1)) \rhd [i^{s_0}?1].nil)/\mathcal{L}(s_1, s_2)\end{aligned}$$

11. Store the value of $e$ in the memory:

$$\mathcal{P}_{11}(s_1, s_2) \mid \mathcal{E}_3 \xrightarrow{\tau} \mathcal{P}_{12}(s_1, s_2) \mid \mathcal{E}_4$$

where:

$$\begin{aligned}\mathcal{P}_{12}(s_1, s_2) \;=\; & (((S_4^a(7) \mid S_4^b(14)) \rhd [i^{s_1}?1].\Phi(s_1) \mid (S_4^e(15) \mid O_4(e) \mid \\ & R_3(B_0, s_0, click)) \rhd [i^{s_1}?1].\Phi(s_1)) \rhd [i^{s_0}?1].nil)/\mathcal{L}(s_1, s_2)\end{aligned}$$

12. Output the value of $e$:

$$\mathcal{P}_{11}(s_1, s_2) \mid \mathcal{E}_3 \xrightarrow{\tau} \mathcal{P}_{12}(s_1, s_2) \mid \mathcal{E}_5$$

where:

$$\begin{aligned}\mathcal{P}_{12}(s_1, s_2) \;=\; & (((S_4^a(7) \mid S_4^b(14)) \rhd [i^{s_1}?1].\Phi(s_1) \mid (S_4^e(15) \mid \\ & R_3(B_0, s_0, click)) \rhd [i^{s_1}?1].\Phi(s_1)) \rhd [i^{s_0}?1].nil)/\mathcal{L}(s_1, s_2)\end{aligned}$$

13. The end-user pushes $B_0$ to exit the program:

$$\begin{aligned}\mathcal{P}_{12}(s_1, s_2) \mid \mathcal{E}_5 \;\xrightarrow{\tau}\; & (((S_4^a(7) \mid S_4^b(14)) \rhd [i^{s_1}?1].\Phi(s_1) \mid (S_4^e(15) \mid \\ & [i^{s_0}?1].nil) \rhd [i^{s_1}?1].\Phi(s_1)) \rhd [i^{s_0}?1].nil)/\mathcal{L}(s_1, s_2) \\ \xrightarrow{\tau}\; & nil\end{aligned}$$

【書類名】外国語図面

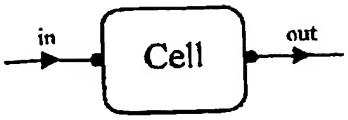



Figure 1: Cell formalized as a process.

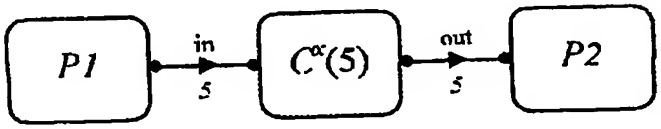



Figure 2: Example of interacting processes.

Table 2: Semantics of Lyee-Calculus

$$(R_{\leadsto})\ \frac{P \xrightarrow{\kappa} Q' \quad Q' \leadsto Q}{P \xrightarrow{\kappa} Q}$$

$$(R_!)\ \frac{v \in [\![e]\!]}{[\imath!e].P \xrightarrow{\imath!v} P} \qquad (R_?)\ \frac{v \in [\![e]\!]}{[\imath?e].P \xrightarrow{\imath?v} P[v/e]} \qquad (R_\tau)\ \frac{v \in [\![e]\!]}{[\tau].P \xrightarrow{\tau} P}$$

$$(R_{[]})\ \frac{[K_1].P \xrightarrow{\kappa} P'}{[K_1 \cup K_2].P \xrightarrow{\kappa} [K_2]P'}\ K_2 \neq \emptyset$$

$$(R^l_+)\ \frac{P \xrightarrow{\kappa} P'}{P+Q \xrightarrow{\kappa} P'} \qquad (R^r_+)\ \frac{Q \xrightarrow{\kappa} Q'}{P+Q \xrightarrow{\kappa} Q'}$$

$$(R^\tau_\triangleright)\ \frac{P \xrightarrow{\imath?v} P' \quad Q \xrightarrow{\imath!v} Q'}{P \triangleright Q \xrightarrow{\tau} Q'} \qquad (R^l_\triangleright)\ \frac{P \xrightarrow{\kappa} P'}{P \triangleright Q \xrightarrow{\kappa} P' \triangleright Q} \qquad (R^r_\triangleright)\ \frac{Q \xrightarrow{\kappa} Q'}{P \triangleright Q \xrightarrow{\kappa} Q'}$$

$$(R^\tau_|)\ \frac{P \xrightarrow{\imath?v} P' \quad Q \xrightarrow{\imath!v} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad (R^l_|)\ \frac{P \xrightarrow{\kappa} P'}{P \mid Q \xrightarrow{\kappa} P' \mid Q} \qquad (R^r_|)\ \frac{Q \xrightarrow{\kappa} Q'}{P \mid Q \xrightarrow{\kappa} P \mid Q'}$$

$$(R_=)\ \frac{P[\vec{Y}/\vec{X}] \xrightarrow{\kappa} P'}{A(\vec{Y}) \xrightarrow{\kappa} P'}\ A(\vec{X}) = P \qquad (R_/)\ \frac{P \xrightarrow{\kappa} P'}{P/L \xrightarrow{\kappa} P/L}\ \kappa_1 \in L$$

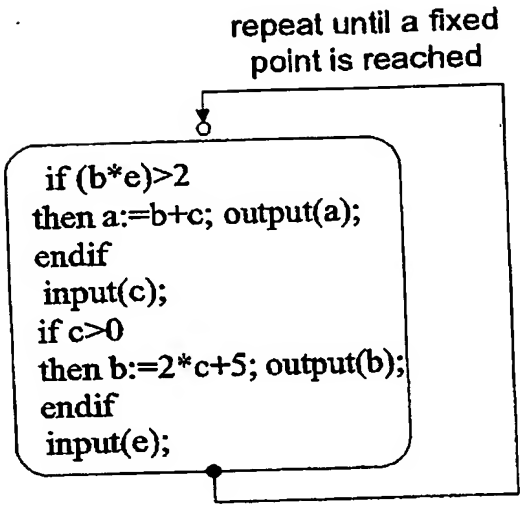



Figure 3: Requirement Execution.

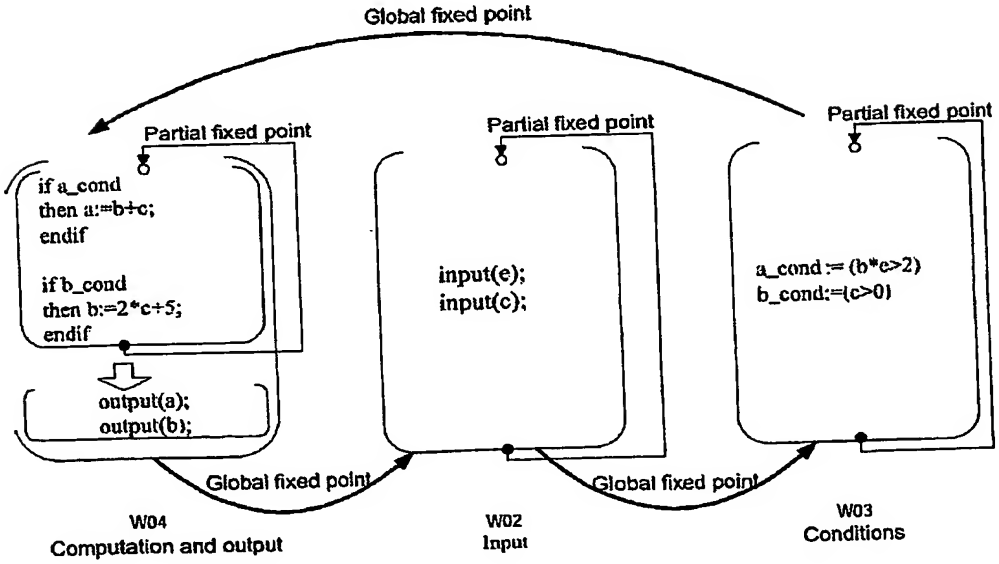



Figure 4: Lyee Pallets.

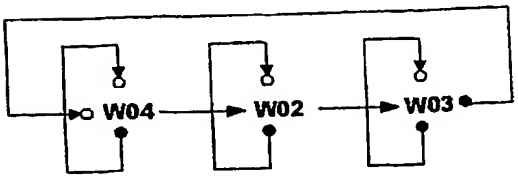



Figure 5: Scenario Function.

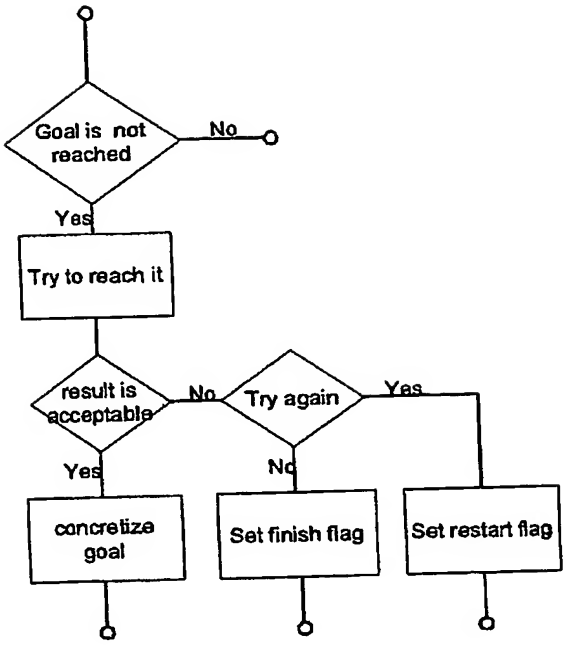
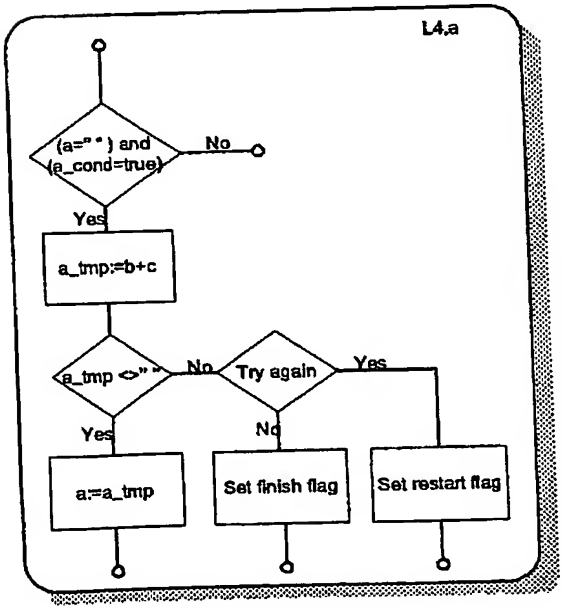
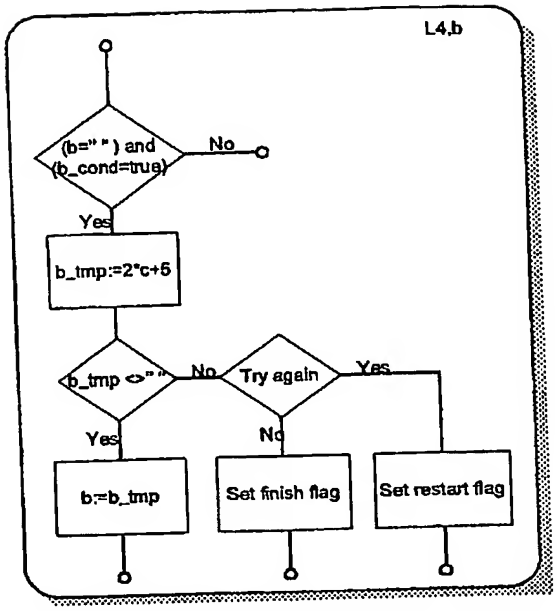
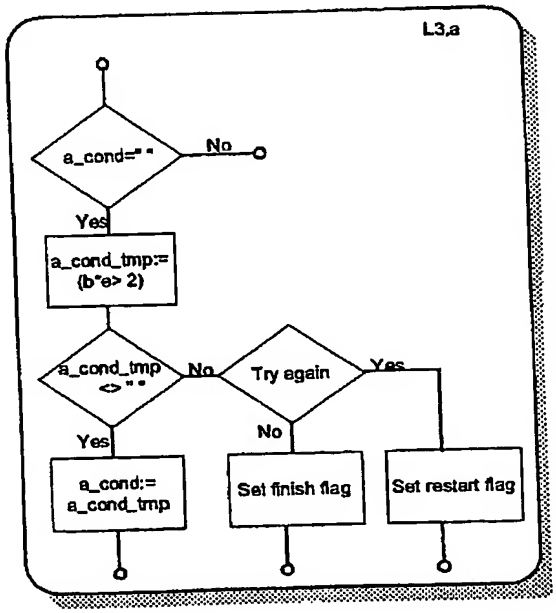
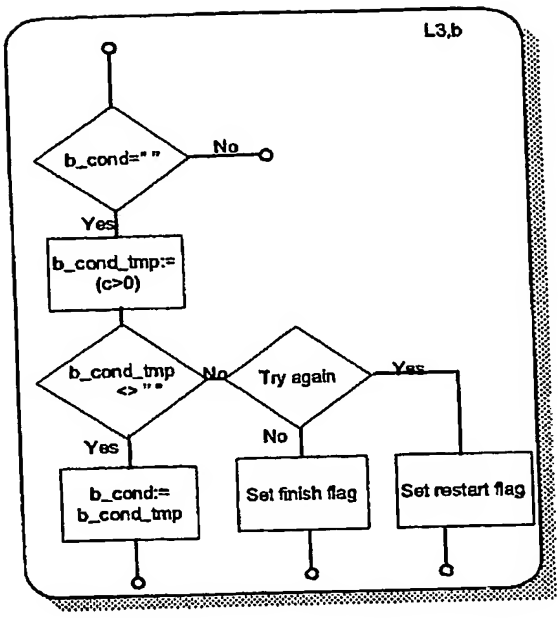



Figure 6: Predicate Vector.




L4,b
(b=" ") and (b_cond=true)
No
Yes
b_tmp:=2*c+5
b_tmp <>" "
No
Try again
Yes
Yes
No
b:=b_tmp
Set finish flag
Set restart flag


L3,a
a_cond=" "
No
Yes
a_cond_tmp:= (b*e> 2)
a_cond_tmp <>" "
No
Try again
Yes
Yes
No
a_cond:= a_cond_tmp
Set finish flag
Set restart flag



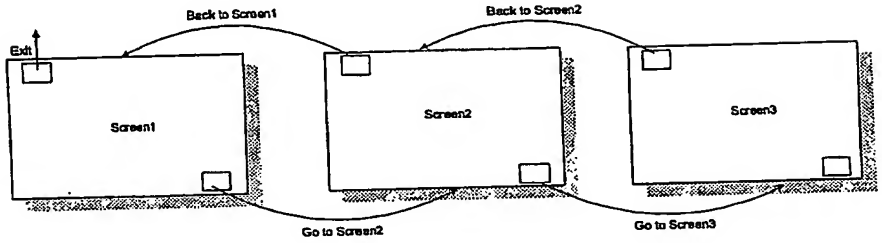




Figure 9: Screen Interactions.

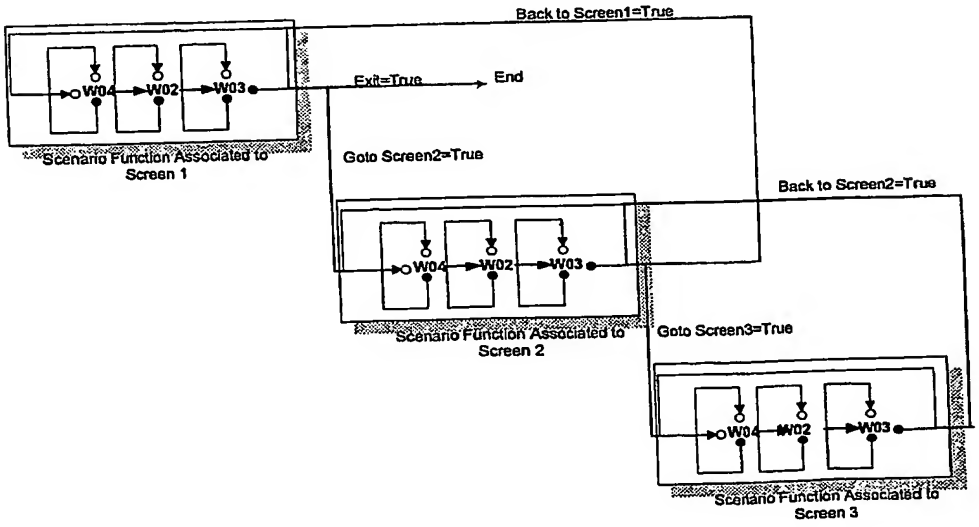



Figure 10: Processes Route Diagram.

**Signification Vectors**

| Vector | Comments |
|---|---|
| $L_4(x,e)$ = | |
| $[\imath_3^x?1].$ | wait to receive the value true (1) on the channel $\imath_3^x$ (from $L_3$) |
| $[\mathcal{F}(Use(e))].$ | wait to receive the value of all the $Use$ of $e$ from the memory |
| $[\jmath_4^x!e].$ | evaluate e and put the result in the memory cell of $x$ |
| $nil$ | finish |
| $L_3(x,c)$ = | |
| $[\mathcal{F}(Use(c))].$ | wait to receive the value of all the $Use$ of $c$ from the memory |
| $[\imath_3^x!c].$ | evaluate $c$ and send the result on the channel $\imath_3^x$ (to $L_4$) |
| $nil$ | finish |
| $L_2(x)$ = | |
| $[\imath_2^x?x].$ | wait to receive the value $x$ on the channel $\imath_2^x$ (from the input vector) |
| $[\jmath_4^x!x].$ | save the value of $x$ in the memory cell of $x$ |
| $nil$ | finish |

**Action Vectors**

| Input Vector | Comments |
|---|---|
| $I_2(x)$ = | |
| $[d_x?x].$ | wait to receive a value on the input channel $d_x$ |
| $[\imath_2^x!x].$ | send the value of $x$ on the channel $\imath_2^x$ (to $L_2$) |
| $nil$ | finish |
| **Output Vector** | **Comments** |
| $O_4(x)$ = | |
| $[\imath_4^x?x].$ | wait to receive the value of $x$ from the memory |
| $[d_x!x].$ | send the value of $x$ on the output channel $d_x$ |
| $nil$ | finish |
| **Memory (Structural Vector)** | **Comments** |
| $S_4(x)$ = $[\jmath_4^x?y].S_4^x(y)$ | wait to receive any value (initialization) and then behave as $S_4^x(y)$ |
| $S_4^x(y)$ = $[\jmath_4^x?z].S_4^x(z)+$ | if you receive $z$ behaves as $S_4^x(z)$ (change the content of the cell $x$) |
| $[\imath_4^x!y].S_4^x(y)$ | if you send $y$ behaves as $S_4^x(y)$ (do not change the content of the cell $x$) |
| **Routing Vector** | **Comments** |
| $R_3(b,e,s)$ = | |
| $[d_b?clik].$ | wait until the bouton $b$ be pushed |
| $[\mathcal{F}(Use(e))].$ | evaluate the condition of the bouton |
| $[\imath^s!e].$ | send the result to the screen $s$ (i.e., activate $s$ if $e=1$) |
| $nil$ | finish |

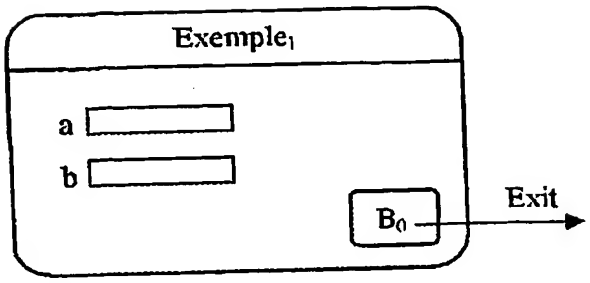



Figure 11: Case Study - One Screen.

| $W_{02}(s_1) = L_2(a) \mid I_2(a)$ | |
|---|---|
| $L_2(a) = [i_2^a?a].$ $[j_4^a!a].$ $nil$ | $I_2(a) = [d_a?a].$ $[i_2^a!a].$ $nil$ |

| $W_{03}(s_1) = L_3(b, a > 0) \mid R_3(B_0, click, s_0)$ | |
|---|---|
| $L_3(b, a > 0) = [i_4^a?a].$ $[i_3^b!(a > 0)].$ $nil$ | $R_3(B_0, click, s_0) = [d_{B_0}?click].$ $[i^{s_0}!1].$ $nil$ |

| $W_{04}(s_1) = S_4(a) \mid S_4(b) \mid L_4(b, 2*a) \mid O_4(b)$ | | |
|---|---|---|
| $S_4(a) = [j_4^a?a].$ $S_4^a(a)$ $S_4(b) = [j_4^b?b].$ $S_4^b(b)$ | $L_4(b, 2*a) = [i_3^b?1].$ $[i_4^a?a].$ $[j_4^b!(2*a)].$ $nil$ | $O_4(b) = [i_4^b?b].$ $[d_b!b].$ $nil$ |

| Process | Comments |
|---|---|
| $\mathcal{E} = [i^{s_1}!1].\mathcal{E}_1$ | activate the program (screen $s_1$) |
| $\mathcal{E}_1 = [d_a!7].\mathcal{E}_2$ | give the value 7 to the word $a$ |
| $\mathcal{E}_2 = [d_b?x].\mathcal{E}_3$ | get the value of the word $b$ |
| $\mathcal{E}_3 = [d_{B_0}!click].nil$ | clicke the bouton $B_0$ to exit |

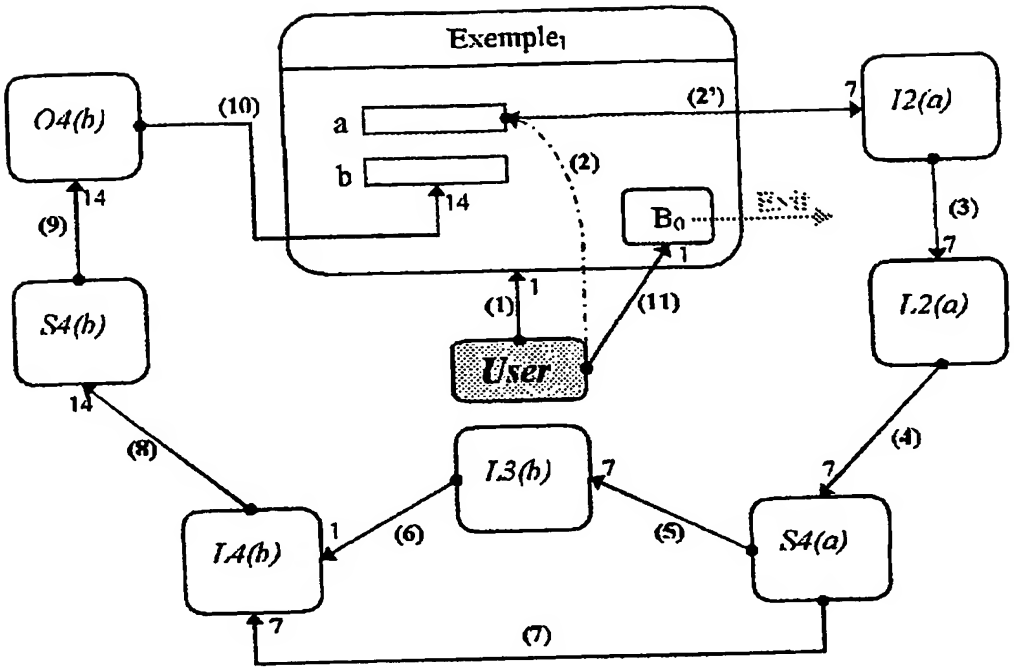



Figure 12: Interaction between processes.

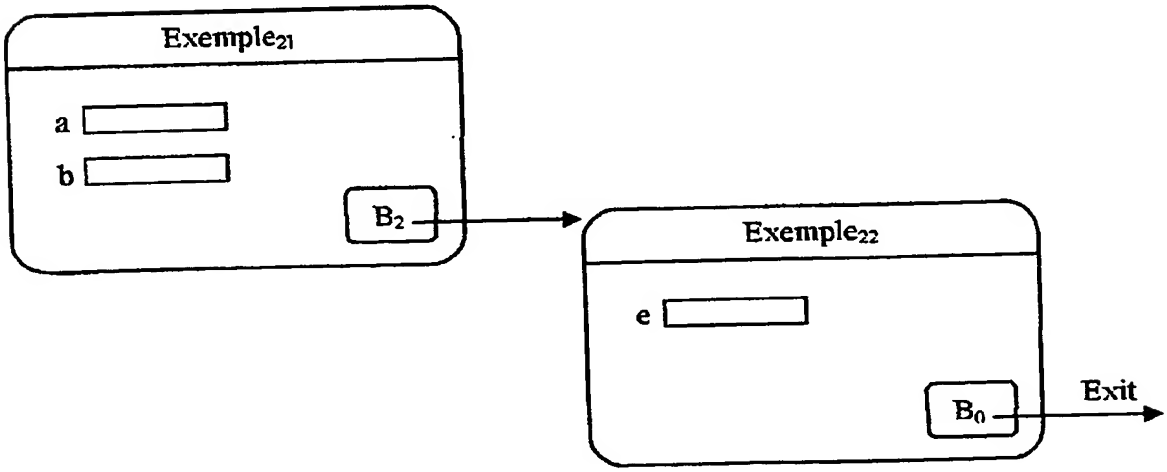



Figure 13: Case Study - Two Screen.

| $W_{02}(s_1) = L_2(a) \mid I_2(a)$ | |
|---|---|
| $L_2(a) = [i_2^a?a].$ $[i_4^a!a].$ $nil$ | $I_2(a) = [d_a?a].$ $[i_2^a!a].$ $nil$ |

| $W_{03}(s_1) = L_3(b, a > 0) \mid R_3(B_2, click, s_2)$ | |
|---|---|
| $L_3(b, a > 0) = [\imath_4^a?a].$<br>$[\imath_3^b!(a > 0)].$<br>$nil$ | $R_3(B_2, click, s_2) = [d_{B_2}?click].$<br>$[\imath^{s_2}!1].$<br>$nil$ |

| $W_{03}(s_2) = L_3(e, b > 0) \mid R_3(B_0, click, s_0)$ | |
|---|---|
| $L_3(e, b > 0) = [\imath_4^b?b].$<br>$[\imath_3^e!(b > 0)].$<br>$nil$ | $R_3(B_0, click, s_0) = [d_{B_0}?click].$<br>$[\imath^{s_0}!1].$<br>$nil$ |

| $W_{04}(s_1) = S_4(a) \mid S_4(b) \mid L_4(b, 2 * a) \mid O_4(b)$ | | |
|---|---|---|
| $S_4(a) = [\jmath_4^a?a].$<br>$S_4^a(y)$<br>$S_4(b) = [\jmath_4^b?b].$<br>$S_4^b(y)$ | $L_4(b, 2 * a) = [\imath_3^b!1].$<br>$[\imath_4^a?a].$<br>$[\jmath_4^b!(2 * a)].$<br>$nil$ | $O_4(b) = [\imath_4^b?b].$<br>$[d_b!b].$<br>$nil$ |
| $W_{04}(s_2) = S_4(e) \mid L_4(e, 1 + b) \mid O_4(e)$ | | |
| $S_4(e) = [\jmath_4^e?e].$<br>$S_4^e(y)$ | $L_4(e, 1 + b) = [\imath_3^e!1].$<br>$[\imath_4^b?b].$<br>$[\jmath_4^e!(1 + b)].$<br>$nil$ | $O_4(e) = [\imath_4^e?e].$<br>$[d_e!e].$<br>$nil$ |

| End-user Action | Comments |
|---|---|
| $\mathcal{E} = [\imath^{s_1}!1].\mathcal{E}_1$ | activate the program |
| $\mathcal{E}_1 = [d_a!7].\mathcal{E}_2$ | give the value 7 to word $a$ |
| $\mathcal{E}_2 = [d_b?b].\mathcal{E}_3$ | wait for the value of the word $b$ |
| $\mathcal{E}_3 = [\imath^{B_2}!click].\mathcal{E}_4$ | pushes the botom $B_2$ to go screen 2 |
| $\mathcal{E}_4 = [d_e?e].\mathcal{E}_5$ | wait for the value of the word $e$ |
| $\mathcal{E}_5 = [\imath^{B_0}!click].nil$ | push the botom $B_0$ to exit |

【書類名】外国語要約書

**Abstract.** Over the last years, various methodologies and techniques have been elaborated and proposed to improve one or many aspects related to the software development life cycle. However, despite the great effort in this research field, the production of clearly understood and modifiable systems still an ambitious goal and far from reached. This is due, on one hand, to the complexity and the subtlety of software themselves and, on the other hand, to the limitations of the current methodologies. Recently, a new and very promising methodology, called Lyee, has been proposed. Intended to deal efficiently with a wide range of software problems related to different field, Lyee allows the development of software by simply defining their requirements.

Nevertheless, since both the semantics of Lyee generated software together with the process of automatic generation of software from requirements are described using informal language, difficulties and confusions may arise when trying to understand and study this methodology.

The main purpose of this paper is to formalize, using a process algebra, the process of automatic generation of software together with the semantics of Lyee generated software. Actually, process algebra naturally support many concept of the Lyee methodology and consequently formalize them simply and elegantly. It offers the Lyee methodology an abstract machine more suitable than the Von-Newman one. In fact, this new abstract machine consider a program as chemical solution when molecules (different vectors of the lyee methodology) interact together to reach a collective goal.

【書類名】　出願人名義変更届
【提出日】　平成16年 8月 2日
【あて先】　特許庁長官　殿
【事件の表示】
　【出願番号】　特願2003-330772
【承継人】
　【識別番号】　396023362
　【住所又は居所】　東京都江東区潮見２丁目１０番２４号
　【氏名又は名称】　カテナ株式会社
　【代表者】　小宮　善継
【提出物件の目録】
　【物件名】　権利の承継を証明する書面　1

【物件名】　権利の承継を証明する書面

【添付書類】

平成16年7月30日

# 持分譲渡証書

住所　東京都江東区潮見２丁目１０番２４号
譲受人　カテナ株式会社
代表取締役　小宮　善継　殿

住所　東京都港区高輪３丁目１１番３号
譲渡人　ソフトウェア生産技術研究所株式会社
代表取締役　小宮　善継

下記の発明に関する特許を受ける権利については弊社と貴社との共有のところ、今般、弊社の持分を貴社に譲渡したことに相違ありません。

記

１．特許出願番号　特願２００３－３３０７７２

２．発明の名称　ソフトウェア生成方法

# 認定・付加情報

| | |
|---|---|
| 特許出願の番号 | 特願２００３－３３０７７２ |
| 受付番号 | １０４０１４４００２１ |
| 書類名 | 出願人名義変更届 |
| 担当官 | 福田　政美　　７６６９ |
| 作成日 | 平成１６年　９月１０日 |

＜認定情報・付加情報＞

【提出された物件の記事】

【提出物件名】　権利の承継を証明する書面　1

特願２００３－３３０７７２

出　願　人　履　歴　情　報

識別番号　　［３９６０２３３６２］

１．変更年月日　　１９９６年１０月１６日
［変更理由］　　新規登録
住　所　　東京都江東区潮見二丁目１０番２４号
氏　名　　カテナ株式会社

特願２００３－３３０７７２

出　願　人　履　歴　情　報

識別番号　[５９９０８６２３８]

１．変更年月日　１９９９年　６月２１日
[変更理由]　新規登録
住　所　東京都港区高輪３丁目１１番３号
氏　名　ソフトウェア生産技術研究所株式会社